

Ome Srirama

Chair of Information Systems
LuFG Cooperation Systems
Aachen University of Technology
Prof. Dr. Wolfgang Prinz

Master Thesis

Concept, implementation and performance testing of a mobile Web Service provider for Smart Phones

of

Satish Narayana Srirama

Matriculation Number: 247512

Aachen, 26th July 2004

Supervised by:

Prof. Dr. Wolfgang Prinz
Dipl.-Ing. Martin Gerdes (EED Research)

I assure, that this work has been done solely by me without any further help from others except for official attendance by the Chair of Information Systems. The literature used in the study is listed completely in the bibliography.

Aachen,

(Satish Narayana Srirama)

To my beloved father, Dr. Lakshminarayana Srirama,
and mother, Lolakshi Srirama.

SSN

ABSTRACT

Web Services are loosely coupled, standard-based, reusable, distributed software components that are programmatically accessible over standard Internet protocols. Web Services range from simple requests, such as stock quotes or user authentication, to more complex tasks, such as comparing and purchasing items over the Internet.

With the 2.5G (Interim Generation) mobile communication technologies in the cellular domain like GPRS/EDGE, the speed of wireless data transmission have increased significantly, which gives a large scope for the Web Services even in the cellular domain and thereby enabling the implementation of distributed applications even on Smart Phones.

This thesis studies and discusses the usage and feasibility of mobile terminals in the Web Services domain as both Web Service requestors and Web Service providers in detail, with main focus on “Mobile Terminals as Web Service providers”.

For the study of mobile terminals as Web Service requestors, the mobile terminal, which uses SOAP over HTTP for the service access, is observed and compared for performance issues with different other standard distributed protocols like Java RMI¹.

Similarly for the study of mobile terminals as Web Service providers, a standard Web Service provider was developed for Smart Phones, upon which different Web Services can be deployed. Once such a Web Service provider is deployed on the Smart Phone, the means of identification and addressing of the deployed services on the mobile Web Service provider, that allow the Web Services to be accessible also from outside the mobile network operator’s network domain, are studied in detail. The thesis also discusses the applications of such a mobile Web Service provider, and some example Web Services were developed to prove the feasibility of such a “Mobile Host”.

Furthermore, the study addresses different performance issues of such a mobile Web Service provider developed for Smart Phones, like total processing time, transmission time, memory load etc., under both the single and concurrent accesses of the Web Service clients.

As a whole the study shows that the Smart Phones can be used in Web Services domain as both Web Service providers and clients. Finally, the thesis discusses some of the areas where there is scope for further research in the mobile Web Services domain.

¹ Sun TM’s Java Remote Method Invocation (RMI). More details can be found at <http://java.sun.com>

Table of Contents

ABSTRACT	2
1 INTRODUCTION	6
1.1 MOTIVATION	6
1.2 OVERVIEW	6
2 STATE OF THE ART	8
2.1 WEB SERVICES	8
2.1.1 <i>Web Services overview</i>	8
2.1.2 <i>WS Architecture</i>	9
2.1.2.1 WS Components	10
2.1.2.2 WS Operations	10
2.1.2.3 SOAP - Simple Object Access Protocol	11
2.1.2.4 WSDL - Web Services Description Language	16
2.1.2.5 UDDI - Universal Description, Discovery and Integration	19
2.2 WIRELESS TECHNOLOGIES	21
2.2.1 <i>HSCSD</i>	22
2.2.2 <i>GPRS</i>	22
2.3 CURRENT STATUS OF MOBILE WEB SERVICES FOR SMART PHONES	23
2.4 KSOAP – LIGHTWEIGHT SOAP TOOLKIT	23
3 ARCHITECTURES	28
3.1 MOBILE TERMINAL AS WEB SERVICE REQUESTOR	28
3.2 MOBILE TERMINAL AS WEB SERVICE PROVIDER (“MOBILE HOST”)	29
3.2.1 <i>Basic architectural setup</i>	29
3.2.2 <i>Mobile terminal access</i>	30
3.2.2.1 HSCSD	30
3.2.2.2 GPRS	31
3.2.3 <i>Alternative scenarios</i>	33
3.2.3.1 Virtual mobile Web Service provider	33
3.2.3.2 Searching for IP at NAT	35
3.2.3.3 Workaround with session maintenance	35
4 POSSIBLE APPLICATIONS WITH MOBILE WEB SERVICE PROVIDERS	38
4.1 MOBILE PHOTO ALBUM SERVICE	38
4.2 LOCATION (GPS) DATA PROVISIONING	39
4.3 MOBILE GAMING	40
4.4 TRANSPORTATION AND LOGISTICS	41
5 “SSNSERVER” – MOBILE WEB SERVICE PROVIDER	46
5.1 TYPES OF JAVA FOR MOBILE PHONES	46
5.1.1 <i>PersonalJava</i>	46
5.1.2 <i>J2ME – Java 2 Platform, Micro Edition</i>	47
5.2 HTTP (HYPERTEXT TRANSFER PROTOCOL)	48
5.2.1 <i>HTTP message format</i>	49
5.2.1.1 HTTP Request message	49
5.2.1.2 HTTP Response message	50
5.3 ARCHITECTURE AND FEATURES OF THE MOBILE HOST	51

1. Introduction

5.4	IMPLEMENTATION DETAILS	55
5.4.1	General features of the Mobile Host.....	55
5.4.2	Package hierarchy	55
5.4.3	Mobile Web Service provider	56
5.4.4	Services developed	58
5.4.5	Mobile Host GUI	61
6	PERFORMANCE ANALYSIS.....	64
6.1	MOBILE TERMINAL AS WEB SERVICE CLIENT.....	64
6.1.1	Test setup.....	64
6.1.2	Test cases.....	65
6.1.3	Experimental results	65
6.1.3.1	Evaluation of the exchanged data	66
6.1.3.2	Response times	68
6.1.4	Observations.....	68
6.2	MOBILE TERMINAL AS WEB SERVICE PROVIDER	69
6.2.1	Test setup.....	69
6.2.2	Traces architecture.....	69
6.2.3	Experiments & results	71
6.2.4	Observations.....	84
7	FUTURE RESEARCH DIRECTIONS	86
7.1	PROXY ARCHITECTURE	86
7.2	BEEP FOR SOAP MESSAGE TRANSMISSION	86
8	CONCLUSION	88
	LIST OF FIGURES	90
	LIST OF TABLES	92
	APPENDIX A - TEST CASE IMAGES	94
	APPENDIX B – SOME FACTS AND FINDINGS	96
	BIBLIOGRAPHY	98

1 Introduction

1.1 Motivation

Traditionally, Internet servers - particularly Web servers - mainly serve static content. Since quite some time now, the development goes into the direction of dynamic content, thus enabling personalization of Web pages, as well as more complex interaction as required for example for on-line commerce and electronic banking. The latest trends in the field of Web interaction are Web Services. Web Services are software components that can be accessed over the Internet using established Web mechanisms and protocols such as SOAP and HTTP. Public interfaces of Web Services are defined and described using Extensible Markup Language (XML) based definitions. Examples of Web Services range from simple requests, such as stock quotes or user authentication, to more complex tasks, such as comparing and purchasing items over the Internet. Powerful integrated development environments allow an easy, fast and flexible development and deployment process for enhanced applications based on physically and logically distributed network resources.

With the 2.5G (Interim Generation) mobile communication technologies in the cellular domain like GPRS/EDGE, the speed of wireless data transmission has increased significantly, reaching up to 144 kbps, thereby enabling better applications and usage of mobile devices in different application domains.

Based on these developments it is a logical next step to turn mobile devices into wireless Web Service requestors and even providers, and by this enabling communication via open XML Web Service interfaces and standardized protocols also on the radio link, where today still proprietary and application- and terminal-specific interfaces are required. This leads to manifold opportunities to mobile operators, wireless equipment vendors, third-party application developers, and the end users. It is easy to imagine that in the future mobile applications based on Web Service clients will generate a large percentage of all Web Service requests, and on top of that mobile devices will even be used as Web Service providers in specific application areas.

1.2 Overview

The main objective of the thesis is to study the usage and feasibility of mobile terminals in the Web Services domain as both service clients and service providers, with main focus on “Mobile Terminals as Web Service providers”. With this intention, a rudimentary Web Service provider was developed for Smart Phones and thoroughly tested for the performance issues. The results have proved the feasibility of a Web Service provider on a (high end) mobile terminal.

With the intent described above the thesis has been accomplished and reported in the following chapters:

Chapter 2: State of the art

The chapter discusses the state of the art for the thesis. It first discusses generally Web Services technologies and standards like SOAP, WSDL etc., and then briefly the current wireless technologies used in the study. The chapter also discusses the current status of mobile Web Services and available lightweight SOAP parsers.

1. Introduction

Chapter 3: Architectures

The chapter describes the basic architectures of the mobile Web Services, with the mobile terminal as both Web Service provider and Web Service client. The chapter then describes different methods and architectures for identifying and addressing the Web Services deployed on the mobile Web Service provider. Apart from these, the chapter also discusses alternative scenarios considered during the study.

Chapter 4: Possible applications with mobile Web Service providers

The chapter discusses some of the numerous possible applications utilizing mobile Web Service providers deployed on Smart Phones.

Chapter 5: “SSNServer” – Mobile Web Service Provider

The chapter describes the used design architectures and technologies, and explains the implementation details of the “SSNServer”, i.e. the mobile Web Service provider that has been developed.

Chapter 6: Performance analysis

The chapter first gives a description of the performance analysis conducted in the mobile Web Service environment with mobile terminals as Web Service clients and providers, and then explains the experiments conducted and the analyzed results in detail.

Chapter 7: Future research directions

The chapter gives a brief description of different areas opening further research potential.

Chapter 8: Conclusion

The chapter summarizes the results of the work for this thesis.

2 State of the art

The following chapter discusses the state of the art forming the basis for this thesis. The chapter first discusses Web Services technologies, and then briefly explains the current wireless technologies used in the study. The chapter also discusses the current status of mobile Web Services and available lightweight SOAP parsers.

2.1 Web Services

The following sub chapter gives overview of the Web Services technology and standards like SOAP, WSDL and UDDI.

2.1.1 Web Services overview

“Loosely coupled, standard-based reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard Internet protocols.” [19]

A Web Service is a software function, identified by a URI, whose public interfaces and bindings are defined and described using WSDL¹ (Web Services Description Language, based on XML²(Extensible Markup Language)). The definition of a Web Service can be exported to a file, published to a lookup service, and discovered by other software systems. These systems may then interact with the Web Service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.

The Web Service architecture defined by the W3C enables application-to-application communication over the Internet. Web Services allow access to software components through standard Web technologies, regardless of platforms, implementation languages, etc.

Hence, Web Services are self-contained, modular applications that can be:

- Described using a service description language, such as the Web Services Description Language.
- Published by registering descriptions and access policies with a well-known registry like UDDI³ (Universal Description, Discovery and Integration) registry.
- Found by sending queries to that registry and receiving the binding details of the service(s) that fit the parameters of the query.
- Bound by using the information contained in the service description to create a callable service instance or proxy.
- Invoked over a network by using the information contained in the binding details of the service description.
- Composed with other services into new services and applications.

¹ WSDL - specification available at <http://www.w3.org/TR/wsdl>

² XML - specification available at <http://www.w3.org/XML/>

³ UDDI - specification available at <http://www.uddi.org/>

2. State of the art

Service providers deploy Web Services on the Internet. The functions provided by a given Web Service are described using the Web Services Description Language (WSDL). Service providers can publish deployed services on the Web. A *service broker* can help service providers and service requestors to “find each other” and set up a business relationship. Therefore the service requestor can use an API to ask the service broker about the services it needs. When the service broker returns results (that means available service descriptions), the service requestor can use those results to bind to a particular service.

All of the communications we've discussed here can take place over SOAP¹ (Simple Object Access Protocol). SOAP is an XML-based protocol that allows applications to invoke methods on remote objects. Detailed discussion of SOAP is deferred to the later parts of this chapter at section 2.1.2.3.

2.1.2 WS Architecture

The basic architecture for Web Services is built upon its three components: Service Requestor (Client), Service provider and Service Registry. The architecture is shown in Figure 2.1 with its components and the pattern of communication between these components.

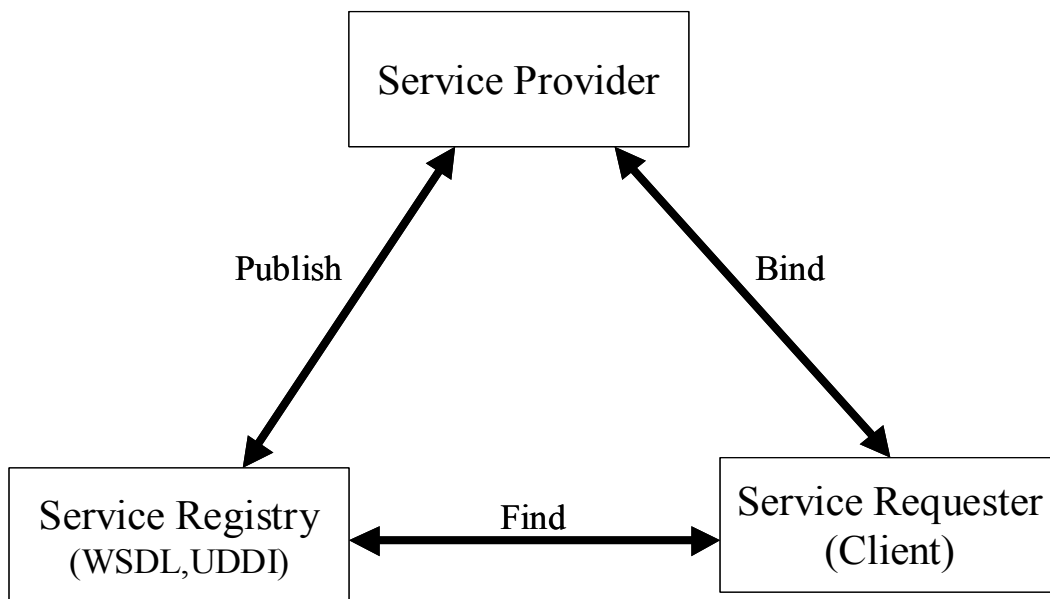


Figure 2.1: Basic operational relationships between Web Service components

The service provider publishes its Web Services in the service registry with the WSDL of the services. The service requestor searches (“Find”) the UDDI registry for the services, and the UDDI compatible service registry returns the respective WSDL. Using the WSDL, the service requestor communicates with the service provider using SOAP for the provided Web Service.

The following sub sections discuss Web Service components and operations, and then explain briefly the standards and protocols involved in this architecture.

¹ SOAP - specification available at <http://www.w3.org/TR/SOAP>

2.1.2.1 WS Components

The following sub section gives a brief description of the Web Service components of the basic Web Service architecture shown in Figure 2.1.

Service:

The service is an application that is provided to a service requestor, who in turn has to fulfill the prerequisites specified by the service provider. Its implementation is deployed on a network-accessible platform. It is described through a service description language. Its description and access policies have been published to a registry.

Service provider:

From a business perspective; it is the owner of the service, and from an architectural perspective; it is the platform that provides access to the service.

Service registry:

The service registry is a searchable repository of service descriptions, where service providers publish their services and service requestors find services and obtain binding information for these services.

Service requestor (Client):

From a business perspective; it is the business that requires certain functions to be fulfilled and from an architectural perspective; it is the application or client that is looking for and invoking a service.

2.1.2.2 WS Operations

The following subsection gives a brief description of the Web Service operations introduced in the basic Web Service architecture shown in Figure 2.1.

Publish / Unpublish:

Service providers advertise (publish) the availability of their service to one or more service registries, or remove (unpublish) their service.

Find:

Service requestors, or service brokers on behalf of service requestors, interact with one or more service registries to discover (find) a set of services that they need for their applications.

Bind:

Service requestors negotiate with service providers to access and invoke services.

Figure 2.2 shows the basic activities performed at the service provider and the service requestor. The activities are shown in the form of activity diagrams for service provider and the service requestor.

2. State of the art

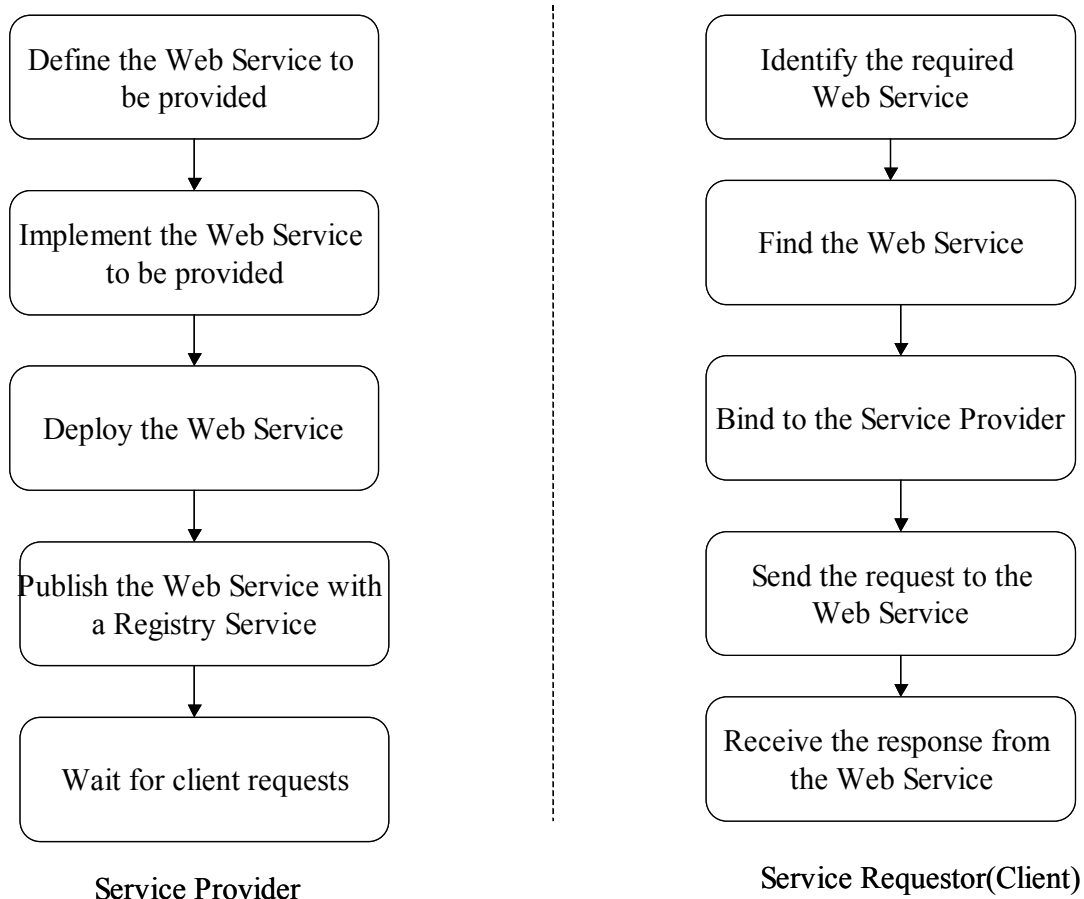


Figure 2.2: Activity diagrams of service provider and service requestor

The following sub sections discuss the standards and protocols involved in this basic Web Services architecture like SOAP, WSDL and UDDI.

2.1.2.3 SOAP - Simple Object Access Protocol

SOAP provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML. It is a simple XML based protocol that lets applications exchange information over different protocols like HTTP¹, FTP², BEEP³, etc. The discussion of SOAP over different protocols is deferred to chapter 7.2.

A SOAP message is an XML document that consists of a mandatory SOAP envelope, an optional SOAP header, and a mandatory SOAP body. The SOAP message structure is shown in Figure 2.3.

¹ HTTP (Hypertext Transfer Protocol), specification available at <http://www.w3.org/Protocols/rfc2616/rfc2616.html>

² FTP (File Transfer protocol), specification is available at <http://www.ietf.org/rfc/rfc0959.txt>

³ BEEP (Blocks Extensible Exchange Protocol), specification available at <http://www.ietf.org/rfc/rfc3288.txt>

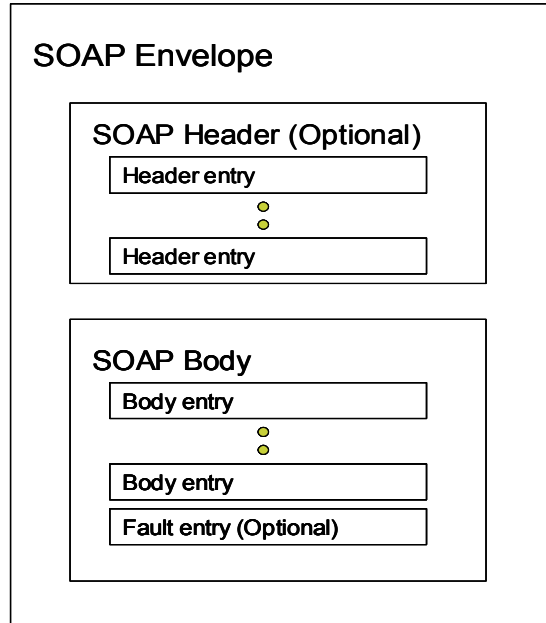


Figure 2.3: SOAP message structure

The Envelope is the root element of the XML document representing the SOAP message. The Header is a generic mechanism for adding features to a SOAP message in a decentralized manner without prior agreement between the communicating parties. SOAP defines a few attributes that can be used to indicate who should deal with a feature and whether the feature is optional or mandatory. The Body is a container for mandatory information intended for the recipient of the SOAP message. SOAP defines an optional element for the body, which is the Fault element used for reporting errors. Each of these parts, of the SOAP message is described in detail in the following sub sections.

The following XML structure of the SOAP 1.1 message gives an idea of the SOAP message.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Header>
    ...
  </SOAP-ENV:Header>

  <SOAP-ENV:Body>
    ...
    <SOAP-ENV:Fault>
      ...
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The SOAP encodingStyle global attribute is used to indicate the serialization rules used in a SOAP message. The children of the SOAP envelope can explicitly override this value. The

2. State of the art

serialization rules are defined in the SOAP specification, which is identified by the URI¹ <http://schemas.xmlsoap.org/soap/encoding/>. The version of the SOAP message can be identified with the value of the namespace. If a different namespace is used, the application must generate an error and discard the message. SOAP version 1.2 uses “<http://www.w3.org/2003/05/soap-envelope>” for namespace and uses “<http://www.w3.org/2003/05/soap-encoding>” for the encodingStyle.

The two versions of SOAP (SOAP 1.1, SOAP 1.2) are almost the same, and the differences between the two versions are available at “<http://www.w3.org/TR/soap12-part0/>”. SOAP 1.1 is widely being used, as of now and for the thesis study, this protocol was used.

SOAP Envelope

The SOAP envelope element is mandatory and it is the root element of the XML based SOAP message. The element itself defines the XML document as the SOAP message. The SOAP envelope contains:

- The ‘Envelope’ element.
- The envelope’s namespace declaration with value <http://schemas.xmlsoap.org/soap/envelope/>.
- The encodingStyle attribute explained earlier.
- It can also have additional attributes like xsd (XML Schema² definition), xsi (XML Schema-instance) etc. If present, such additional attributes must be namespace-qualified³.
- The element may also contain additional sub elements. If present these elements must be namespace-qualified and must follow the SOAP body element.
- An optional Header element.
- A mandatory Body element.

SOAP Header

SOAP provides a flexible mechanism for extending a message in a decentralized and modular way without prior knowledge between the communicating parties. Typical examples that can be incorporated as the header entries are authentication information, transaction management details, payment details etc.

The Header element is optional. The Header element is the first immediate child element of the SOAP envelope element, if at all it exists. All immediate child elements of the Header element are called header entries. The SOAP encodingStyle attribute may be used to indicate the encoding style used for the header entries.

The following example shows a Header for element “Transaction”, including the attribute “mustUnderstand” of value “1”, the attribute “actor” of value <http://mercury.westend.com/wizard/Testbed/>, and the value 2442.

¹ Uniform Resource Identifiers (URI) are short strings that identify resources in the Web like documents, images, downloadable files, services, electronic mailboxes, and other resources.

² More details of XML schema are available at <http://www.w3.org/XML/Schema>

³ More details of XML namespaces and qualified names at <http://www.w3.org/TR/REC-xml-names/>


```

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Header>
  <t:Transaction xmlns:t="http://mercury.westend.com/wizard/SmartServ/"
    SOAP-ENV:mustUnderstand="1"
    xmlns:actor="http://mercury.westend.com/wizard/Testbed/" >
    <t:id> 2442 </t:id>
  </t:Transaction>

</SOAP-ENV:Header>

<SOAP-ENV:Body>
  ...
  ...
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

The SOAP “actor” attribute is used to indicate the recipient of a header element. The value of the SOAP actor attribute is a URI.

A SOAP message travels from the originator to the destination, by passing through different SOAP intermediaries along the message path. A SOAP intermediary is an application that is capable of both receiving and forwarding SOAP messages. The header entry can be for a specific intermediary that can only process the header entry. The “actor” attribute is useful in such a scenario.

The SOAP attribute “mustUnderstand” is used to indicate whether a header entry is mandatory or optional for the recipient to process. The allowed values for the attribute are “0” and “1”, with “0” being default value. A value of “1” indicates that the recipient must understand the header entry. If not, the processing of the message should fail at the recipient.

SOAP Body

The SOAP “Body” element provides a simple mechanism for exchanging the mandatory information intended for the destination recipient of the SOAP message. Generally the Body element contains marshaled RPC calls and error reports.

In the SOAP envelope element, the Body element is mandatory. If a Header element is present, then the Body element must immediately follow the Header element, otherwise it is the first child element of the SOAP envelope element.

All immediate child elements of the Body element are called body entries, and each body entry is encoded as an independent element within the SOAP body element. A body entry is identified by its fully qualified element name, consisting of the namespace URI and the local name. The SOAP “encodingStyle” attribute is used to indicate the encoding style used for the body entries. As explained earlier, the “encodingStyle” attribute can have a different value from the value specified in the SOAP envelope element.

The following example XML code describes the Body element in detail. The message is the SOAP request message for a small service providing the price of an item.

```

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"

```

2. State of the art

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsi="http://www.w3.org/2001/XMLSchema"
>

<SOAP-ENV:Body
  SOAP-ENV:encodingStyle="http://mercury.westend.com/wizard/encoding/">
  <ps:GetPrice xmlns:ps="http://mercury.westend.com/wizard/ws/prices">
    <ps:Item xsi:type="xsd:string"> Sony Ericsson P800 </ps:Item>
  </ps:GetPrice>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The message's Body element specifies that the body entry is the "ps:GetPrice", with a request for the price of the "Item" with value "Sony Ericsson P800". The type of the parameter ("Item") is specified to be string, using the "xsi:type" attribute value "xsd:string". The response of such a SOAP request is as shown in the following example.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema"
>

<SOAP-ENV:Body
  SOAP-ENV:encodingStyle="http://mercury.westend.com/wizard/encoding/">
  <ps:GetPriceResponse xmlns:ps="http://mercury.westend.com/wizard/ws/prices">
    <ps:Price xsi:type="xsd:int"> 583 </ps:Price>
  </ps:GetPrice>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The SOAP response message has a similar structure as the SOAP request message. The Body of the response message shown in the above example returns the price of the Item, which is 583 (Euro, but currency is not defined in this example). The type of the response is specified to be "xsd:int".

The SOAP specification defines one body entry, the "Fault" entry, which is used for reporting errors. More details of the entry are given in the following sub section.

SOAP Fault

An error could occur in processing the SOAP message, at any SOAP intermediary along the message path. The error could occur both during the application processing of the Body element, and at the processing of the SOAP message itself like processing the Header entries.

The SOAP fault element is used to specify the error information within a SOAP message. If a Fault element is present, it must appear as a child element of the Body element. A Fault element can only appear once in a SOAP message.

The SOAP fault element has the following sub elements:

- <faultcode>: The faultcode programmatically identifies the fault. It is mandatory in the Fault entry and its value is a qualified name. The SOAP specification defines some standard fault codes for basic SOAP faults.

- <faultstring>: A human readable explanation of the fault. It is also mandatory in the Fault entry. It gives some information explaining the nature of the fault.
- <faultactor>: The element specifies where (at which SOAP intermediary along the message path) the fault has occurred. The value of the faultactor is a URI.
- <detail>: The detail element is used for carrying application specific error information related to the Body element. It is mandatory when the contents of the Body element could not be processed successfully. It is not used to carry information about error information belonging to header entries. The element can be used to distinguish whether the Body element was processed or not in case of a fault situation since the absence of the detail element in the Fault entry indicates that the fault is not related to processing of the Body element.

As explained earlier, the faultcode identifies the fault. SOAP defines some standard faultcodes, which must be used in the respective situations. The faultcode is a qualified name. The value to the left of the '.' is a more generic fault code value than the value to the right. The standard fault codes are:

- VersionMismatch: when an invalid namespace is found for the SOAP envelope element.
- MustUnderstand: when the immediate child element of the Header element, with the MustUnderstand attribute set to "1", was not understood by the intermediary processing the SOAP message.
- Client: The Client error indicates that the message was incorrectly formed or did not contain the appropriate information for processing the SOAP message. For example, the message could lack the correct authentication information or payment information.
- Server: The Server indicates that the process of the message failed not because of the message details but because of the server processing the SOAP message. The situation could arise in a case where the server requires another processor for processing the message, and the respective processor is unavailable at that particular instance. The processing of the SOAP message could be successful at a later instance of time.

2.1.2.4 WSDL - Web Services Description Language

WSDL is an XML based specification defining how to describe web services. WSDL describes Interface information describing all available public functions, data type information for all message requests and message responses, binding information about the transport protocol to be used and address information for locating the specified Web Service etc.

Using WSDL, a client can locate a Web Service and invoke any of its publicly available functions. The process can also be automated, enabling applications to easily integrate with new services with little or no manual code and interaction.

For defining a standard Web Service, the WSDL specification uses the following six major elements. The elements are shown in Figure 2.4. All of these elements are briefly described below.

2. State of the art

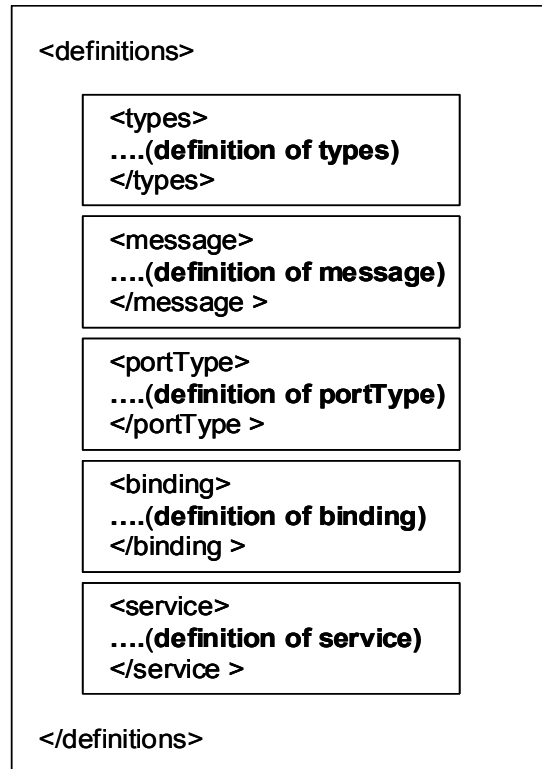


Figure 2.4: The structure of WSDL document

- **<definitions>**: The definitions element must be the root element of all WSDL documents. It defines the name of the Web Service and contains all other service elements described here.
- **<types>**: The types element describes all the data types exchanged between the Web Service requestor and the Web Service provider. It defines the complex data types corresponding to the user defined data types for the services. If the service uses only XML schema built-in simple types, such as strings and integers, the types element is not required.
- **<message>**: The message element defines the input and output parameters of the publicly available functions of a service. It defines the name of the message and contains zero or more message part elements.
- **<portType>**: The portType element is the most important element of the WSDL document. It defines a Web Service, the operations that can be performed, and the messages that are involved for the service.
- **<binding>**: The binding element defines the message format and protocol details for each portType.
- **<service>**: The service element defines the address for invoking the specified Web Service. Generally, this is a URL for the SOAP service being invoked.

The following example WSDL helps in better understanding the above discussed elements. It has been generated for a small calculator service, which has only one method - “add” - that adds the two double numbers specified as the parameters (i1, i2).

We go through the example WSDL element by element. First we discuss the definition of the messages. The WSDL defines two messages for the method “add”; “addRequest” and “addResponse”. The “addRequest” contains two parameters “i1”, “i2” of type “xsd:double” as message parts. The “addResponse” message has only one parameter “addReturn”, also of type “xsd:double”.

```
<wsdl:definitions
targetNamespace="http://localhost:8080/axis/Test/Calculator.jws"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:impl="http://localhost:8080/axis/Test/Calculator.jws"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:message name="addRequest">
    <wsdl:part name="i1" type="xsd:double"/>
    <wsdl:part name="i2" type="xsd:double"/>
  </wsdl:message>
  <wsdl:message name="addResponse">
    <wsdl:part name="addReturn" type="xsd:double"/>
  </wsdl:message>
  ...
  ...
</wsdl:definitions>
```

The WSDL for the service uses a portType named “Calculator”. The portType uses the above defined messages, “addRequest” and “addResponse”, as the input and output messages for the operation. The operation is of pattern type “Request-response”. WSDL supports four basic patterns of operation:

- One-way: The operation can receive a message but will not return a response. The operation therefore has a input message element and no output message element.
- Request-response: The operation receives a message and sends a response. The operation therefore has one input element, followed by one output message element. To encapsulate errors, an optional fault element can also be specified.
- Solicit-response: The operation can send a request and will wait for a response.
- Notification: The operation can send a message, but will not wait for a response

The portType element for the above-described example is given below.

```
<wsdl:portType name="Calculator">
  <wsdl:operation name="add" parameterOrder="i1 i2">
    <wsdl:input message="impl:addRequest" name="addRequest"/>
    <wsdl:output message="impl:addResponse" name="addResponse"/>
  </wsdl:operation>
</wsdl:portType>
```

As discussed earlier, the binding element specifies the protocol details of each portType. Since we are considering mainly Web Services using SOAP in the thesis study, the example uses SOAP binding.

The binding element has two attributes, name and type. The name attribute specifies the name of the binding, and the type attribute specifies the portType being bound. The example uses the above-described portType “impl:Calculator”, where the impl specifies the XML name space to which the portType belongs.

2. State of the art

The (wsdl)soap:binding has two attributes “style” and “transport”. The style attribute can be either “rpc” or “document” indicates whether the operation is a remote procedure call (RPC) or a document-oriented operation.. The example uses “rpc”. The transport attribute specifies the transport protocol used for SOAP. The example uses HTTP to carry SOAP. As stated earlier, the transport protocol can also be FTP or BEEP etc.

The operation element of the WSDL defines each of the operations the port exposes. For each action, the corresponding SOAP action is to be specified¹. The operation also requires how the input and output are encoded. The encoding style used here is “http://schemas.xmlsoap.org/soap/encoding”.

```
<wsdl:binding name="CalculatorSoapBinding" type="impl:Calculator">
  <wsdlsoap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="add">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="addRequest">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://DefaultNamespace" use="encoded"/>
    </wsdl:input>
    <wsdl:output name="addResponse">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://localhost:8080/axis/Test/Calculator.jws"
        use="encoded"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

The service element maps the port (“Calculator”) with binding (“impl:CalculatorSoapBinding”) to a specific address, where the Web Service client can request the Web Service. The service element for the example is shown below.

```
<wsdl:service name="CalculatorService">
  <wsdl:port binding="impl:CalculatorSoapBinding" name="Calculator">
    <wsdlsoap:address
      location="http://localhost:8080/axis/Test/Calculator.jws"/>
  </wsdl:port>
</wsdl:service>
```

2.1.2.5 UDDI - Universal Description, Discovery and Integration

UDDI is a platform-independent framework for describing services, discovering businesses, and integrating business services throughout the Internet. It provides a standardized method for publishing and discovering information about Web Services. It mainly focuses on the process of discovery in the service-oriented architecture, and uses WSDL to describe interfaces to the Web Services.

UDDI provides for several different application areas and use cases, depending on the perspectives and requirements of who is using it. From a business developer's perspective, UDDI is similar to an Internet search engine for business enablers. A business developer can

¹ But for the example described, the WSDL is generated directly by apache Axis, SOAP implementation, which does not use the SOAPAction attribute.

browse one or more UDDI registries to view different businesses that expose Web Services, and the specifications of those services (i.e. the WSDL documents). Software developers can use the “UDDI Programmers API” to query the registry to discover services matching different criteria. Both business developers and software developers can also publish new business entities and services at the UDDI registry.

The information that makes up a registration consists of five data structure types. This separation by information types provides simple partitions to assist in the rapid location and understanding of the different elements of a registration. These data structures, that are passed as input and output parameters of major API messages to and from the UDDI registries, are briefly¹ described below. All these data structures are expressed in XML.

- **Business Entity:** The Business Entity structure represents the basic information of the business. The information includes contact information, categorization, identifiers, descriptions, and relationships to other businesses of the service provider. The UDDI also allows companies to establish relationships with one another. Even in such a case, both the companies should have their respective Business Entity structures. The XML element for this data structure is `<businessEntity>`.
- **Publisher Assertion:** The Publisher Assertion structure is used to establish public relationships between two Business Entity structures. A relationship between two Business Entity structures is visible to the public only when both companies have created the same assertion with two separate Publisher Assertion documents independently. Thus, a company can claim a business relationship only if its partner asserts the same relationship. The XML element for this data structure is `<publisherAssertion>`.
- **Business Service:** A Business Entity contains one or more Business Service structures. A Business Service represents a single, logical service classification. A `<businessService>` element is used to describe a set of services provided by the business. The description of the Business Service includes information like how to bind the Web Service, type of the Web Service etc.
- **Binding Templates:** A Business Service contains one or more Binding Templates. A Binding Template contains the technical descriptions of the Web Services represented by the Business Service structure. It also contains the access point URL of the Web Service, but does not contain the service specification details. It is represented by `<bindingTemplate>`. It is similar to the `<service>` element of the WSDL described earlier.
- **TModels:** A TModel, the `<tModel>` element, is an abstract description of a particular specification or behavior to which the Web Service adheres. For example a TModel can be defined to represent a portType defined by the WSDL. Then a business service implementing the portType can be specified by associating the TModel with one of the binding templates of the business service.

¹ Detailed description of the specification and the data structures at <http://uddi.org/pubs/DataStructure-V2.03-Published-20020719.htm>

2. State of the art

The following Figure 2.5 shows the relationships of the basic UDDI data structures. Detailed description of UDDI and data structures is beyond the scope of this document and can be found at [1].

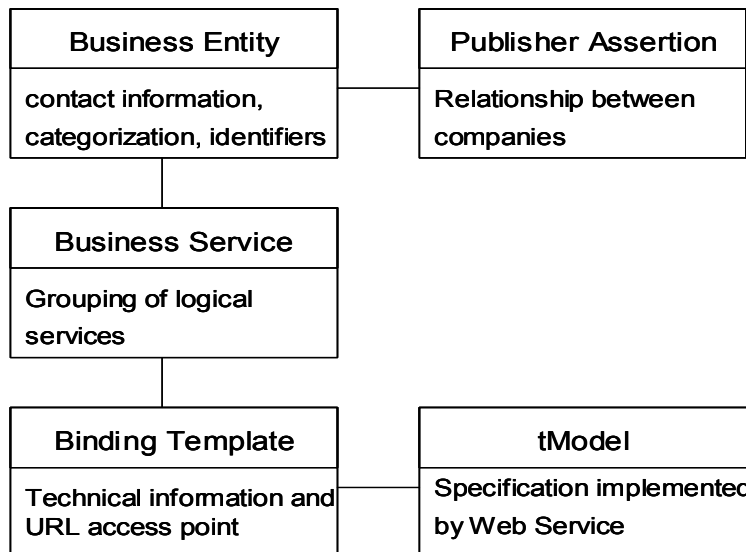


Figure 2.5: Relationships of UDDI data structures

The basic Web Services are implemented as explained in previous subsections. Many Web Services are already being provided on the Internet based on this fundamental architecture. But to make Web Services available to mobile terminals, many new aspects and details have to be considered in the mobile environment.

The following sub chapter gives a brief description of the current wireless technologies and platforms that can be used in the Web Services domain, so that mobile Web Services become a reality.

2.2 Wireless technologies

Today's second-generation GSM¹ networks deliver high quality and secure mobile voice and data services like SMS², circuit switched Internet access etc., with full roaming capabilities and across the world. The GSM platform is a widely successful wireless technology and it is the world's leading mobile standard.

But, with the advent of the 2.5-generation (Interim Generation) technologies like GPRS and EDGE, and 3G technologies like UMTS³, higher transmission rates are achieved in the wireless domain. The following subsections give a brief description of the technologies used for this thesis, like the HSCSD, GPRS etc.

¹ GSM (Global System for Mobile Communications), more details can be found at <http://www.gsmworld.com/index.shtml>

² SMS - Short Message Service

³ UMTS, Universal Mobile Telecommunication System, more details available at <http://www.umtsworld.com/technology/overview.htm>

2.2.1 HSCSD

High Speed Circuit Switched Data (HSCSD)¹ is an enhancement of CSD² (Circuit Switched Data) data services of current GSM networks. HSCSD allows the access of non-voice services with about 3 times higher data rates than CSD. With this technology subscribers are able to send and receive data from their portable computers or mobile devices at a speed of up to 28.8 kbps.

The HSCSD solution enables higher rates by using multiple channels for the data transmission. The HSCSD allows access to company LANs, send and receive e-mails, access the Internet, etc.

2.2.2 GPRS

The General Packet Radio Service (GPRS³) is a non-voice value added service that allows information to be sent and received across a mobile telephone network. The General Packet Radio Service is an extension to second generation GSM. It provides short connection setup times and packet switched connections. GPRS offers faster data transmission via a GSM network within a range of 9.6Kbits to 115Kbits. The available bandwidth can be shared among different users. The high bandwidth is achieved by combining up to eight time slots at the radio interface, where the data is transported in a packet-oriented way.

Since GPRS allows information to be transmitted more quickly, immediately and efficiently across the mobile network, it is a more powerful and less costly mobile data service compared to SMS and Circuit Switched Data. GPRS facilitates instant connections, whereby information can be sent or received immediately as the need arises, subject to radio coverage. No dial-up modem connection is necessary⁴. Hence GPRS users are called as “always connected”. GPRS will fully enable all the Internet applications of the desktop like web browsing, chatting etc. to be feasible over the wireless network.

With the third generation technologies like UMTS (Universal Mobile Telecommunication System), which specifies 3G IP broadband mobile networks that will offer data rates between 156 Kbps and 2Mbps, better transmission rates can be achieved. UMTS networks - and planned service concepts - are currently in the establishment phase.

As the main intension of this study was the Web Service technology and its potential deployment and use cases in available mobile networks, the current established 2.5G technologies, mainly GPRS, are used for the study.

So overall goal of this thesis is to study the Web Services technology as potential solution for mobile applications in current 2.5G GPRS networks.

¹ More details about HSCSD at <http://www.gsmworld.com/technology/hscsd/index.shtml>

² More details about Circuit Switched Data at http://en.wikipedia.org/wiki/Circuit_switching

³ More details about GPRS – General Packet Radio Service at <http://www.gsmworld.com/technology/gprs/index.shtml>

⁴ But (for notebooks e.g.) you still need a wireless modem, that puts your IP connection on top of the GPRS radio service.

2. State of the art

The following subchapter discusses current status of the mobile Web Services domain for the Smart Phones.

2.3 Current status of mobile Web Services for Smart Phones

As far as the Web Service requestors for mobile devices are concerned, quite some research was already done and some standard services are already being provided on the Internet. But the mobile Web Service provision is a totally new concept where much research is yet to be done.

With the intention of finding a lightweight Web Service provider, which could fit on the Smart Phone, extensive search of the Internet led to the conclusion, that not even a basic Web server was available for the Smart Phones to-date, which could be remodeled for the basic Web Service architecture on the mobile terminal. The only server found was a c++ implementation, SmallServ¹ at Symbian, which is not very feasible for the Web Service architecture remodeling because of its proprietary coding.

As a result, a dedicated Web Service provider has been developed for the Smart Phones, to be used for the study of mobile Web Service provision. The details of the mobile Web Service provider are deferred to chapter 5.

The dedicated mobile Web Service provider required means for handling of the SOAP requests from the WS clients. Today there are different parsers available for Java² implementations, from different organization, for processing of SOAP requests, like Xerces³ from Apache, JAXP⁴ from Sun, Xalan⁵ from Apache etc. But, as the service provider was targeted at mobile terminals with limited resources, the parser should have a small memory footprint. With this intent, after an extensive search and analysis of different resources in the WWW, the kSOAP toolkit from enhydra.org was selected.

The following subchapter explains kSOAP, a lightweight SOAP toolkit used in developing the mobile Web Service provider.

2.4 kSOAP – Lightweight SOAP toolkit

kSOAP⁶ is an open source API for SOAP parsing. It is based on kXML. kSOAP provides a SOAP parser with special type mapping and marshalling mechanisms. Both kSOAP and kXML are thin, easy to use, and well documented, and hence they can be used for resource-constrained devices like mobile phones. The kSOAP parser understands the data-type information in SOAP messages and automatically converts the SOAP message to Java data objects, similar to “normal” SOAP parsers. The parser provides programming transparency between a Java program and a SOAP message. A programmer just feeds Java objects into a

¹ SmallServ is a simple Http Server for Symbian OS at <http://www.symbian.com/developer/techlib/apps/smallserv.html>

² The programming language used for the development of mobile Web Service provider, stated earlier, is Java. More details are discussed in the later chapters. Since the implementation was in Java, the thesis considers mostly Java supported tools.

³ More details about Xerces parser at <http://xml.apache.org/xerces2-j/index.html>

⁴ JAXP – Java API for XML Processing, more details at <http://java.sun.com/xml/jaxp/>

⁵ More details about Xalan parser for Java at <http://xml.apache.org/xalan-j/index.html>

⁶ More details about kSOAP at <http://ksoap.enhydra.org/index.html>

SOAP writer, sends the message, waits for the server response, and then reads Java objects directly from the SOAP parser.

The following subsections describe kXML and kSOAP briefly.

kXML

kXML¹ is a lightweight, open source XML parser². Because of its small footprint, it is especially suited for Applets³ or Java applications running on mobile devices. Generally there are two types of parsers in the XML domain for extracting and manipulating the data from the XML documents.

- DOM (Document Object Model): Using DOM mechanism, the whole XML document is parsed into a tree structure, which can be traversed to manipulate and extract the data from the XML document. The DOM provides a rich set of functionality. However, it has some serious limitations, as the whole document should be in memory, which requires significant amount of runtime memory, and the whole document is to be read before constructing the tree, which requires significant amount of time.
- SAX (Simple API for XML): SAX is an event based parsing model. When a XML document is being parsed, the SAX parser generates events, and these events are to be caught and handled by the event listener. The events are generated at different instances like start and end of the document, start and end of an element etc. The parser is also called a “push parser”, as the parser pushes the events to the listener, which catches and handles them. SAX requires less memory.

kXML uses a DOM parser with some modifications. It uses XML pull parser mechanism. Using the pull parser, the application can pull the next event from the parser. When the parser is "pull"-based, the application is in control of, when and where it asks the parser for the next event. The advantage is, that the processing state can be implemented much more natural in local variables and recursions. If we want to parse a small fragment of data, may be at the middle of the document, the data already parsed need not reside in memory, in contrast to “normal” DOM, where all the data resides in memory.

Key features of kXML are[10]:

- XML Namespace support
- "Relaxed" mode for parsing HTML or other SGML formats
- Small Memory footprint
- A Pull-based parser for simplified parsing of nested / modularized XML structures
- XML writing support including namespace handling
- Optional kDOM
- Optional WAP support

kSOAP object structure

In a SOAP message, as discussed earlier, an XML element's xsi:type attribute specifies the data type of the element's content. For example, <myValue xsi:type="xsd:int">123</myValue> specifies an integer value of 123, and <myValue xsi:type="xsd:string">123</myValue> specifies a

¹ More details of kXML and pull parser are available at <http://kxml.enhydra.org/project/aboutProject/index.html>

² An introduction to XML parsers is available at <http://www.javacommerce.com/tutorial/xmldev/>

³ Applet is a Java programmed applications, which can be included in HTML page.

2. State of the art

string value of "123". kSOAP automatically supports the mapping of the following SOAP types to Java types., as shown in Table 2.1.

SOAP type	Java type
xsd:int	java.lang.Integer
xsd:long	java.lang.Long
xsd:string	java.lang.String
xsd:boolean	java.lang.Boolean

Table 2.1: Default data type mapping of kSOAP

When a kSOAP parser encounters an element, the parser reads the XML element into a Java object according to the following rules:

- If the SOAP element is one of the default primitive types as shown in Table 2.1, it is converted to a Java object of a matching type.
- If the SOAP element has no children (a primitive element) but has an unknown primitive type, it is converted to a `SoapPrimitive`¹ object. The element's SOAP type information can be retrieved from the `SoapPrimitive.getNamespace()` and `SoapPrimitive.getName()` methods. The element's String value can be accessed from the `SoapPrimitive.toString()` method.
- If the SOAP element has children (a complex element), it is converted to an object implementing `KvmSerializable` interface. The kSOAP package provides `SoapObject`, as the interface's convenience implementation. Similar to `SoapPrimitive` objects, the element's original SOAP type information can be retrieved from the `SoapObject.getNamespace()` and `SoapObject.getName()` methods. The object holds both user defined and unknown complex types.
- The child elements of a complex element are converted to properties inside the parent's `SoapObject`, according to the rules defined already. Each property also has an associated `PropertyInfo` object containing information such as the SOAP element name and the property's Java object type.

A `SoapObject` can also have other `SoapObjects` as properties. For example, a complex node, which has another complex element as the child, is converted to form such an object. The following Figure 2.6 shows an example Java structure obtained by parsing a SOAP message.

¹ A detailed description of classes and methods of kSOAP API is available as Javadoc at <http://ksoap.enhydra.org/software/documentation/api/>

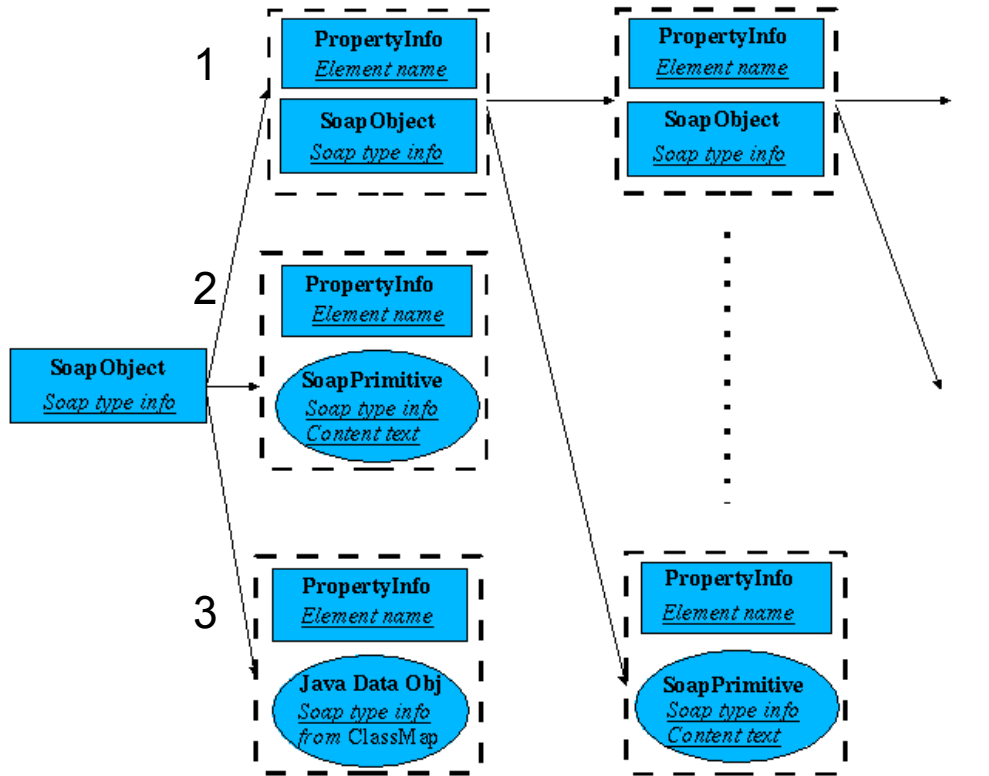


Figure 2.6: The SoapObject structure of a parsed SOAP message[9]

The first property is a complex element, which had another complex element as parameter. So it forms a complex tree structure of the SoapObject. The second property is a SoapPrimitive object. The third property is directly serialized to a Java object, as it is an element of default primitive type.

Marshalling

To automate the SOAP type-mapping process, the following tasks must be prepared for the SOAP parser.

- The SOAP parser must know the mapping relationship between the user defined SOAP types to user defined Java types. This is implemented by adding matching type pairs of SOAP types and Java types to the parser's ClassMap object.
- Since all SOAP types are presented in plain ASCII text strings, the parser must know how to convert the string to a desired Java object. The parser converts the string to a Java object through a user defined marshal object like MarshalDate for java.util.Date, MarshalFloat for java.lang.Float etc., which implements the kSOAP's marshal interface. The marshal object is then registered with the parser's corresponding custom SOAP and Java type pair in the ClassMap object.

The marshal object also defines the rules for serialization and deserialization of the data objects. The marshal interface's writeInstance() and readInstance() methods are overloaded respectively for serializing and deserializing the Java object. More details of the serialization and deserialization are discussed at the implementation details in chapter 5.

2. State of the art

The SOAP types like `xsd:float`, `xsd:base64` etc are all handled with their respective marshal objects.

Summary:

The chapter discussed the state of the art forming the basis of the thesis. The chapter first discussed the Web Services technology; its components, operations and the standards SOAP, WSDL and UDDI. Then it discussed the current wireless technologies used in the study like HSCSD, GPRS etc. The chapter also discussed the current status of mobile Web Services domain and then the kSOAP, a lightweight SOAP toolkit used in developing the mobile Web Service provider.

3 Architectures

This chapter first describes the basic architectures of the mobile Web Services environment, with the mobile terminal as both Web Service provider and Web Service client. Then it describes different methods and architectures for identifying and addressing the Web Services deployed on the mobile Web Service provider. Apart from these, the chapter also discusses alternative¹ scenarios considered during the thesis.

3.1 Mobile terminal as Web Service requestor

The basic architectural setup of Web Services with the mobile terminal as client is shown in the Figure 3.1. In this architecture, the Web Service client is implemented on the mobile terminal. Other than the Web Service requestor, which would be on the mobile terminal, the remaining Web Service components and operations in this architecture are all the same as explained in the basic Web Services architecture in chapter 2.1.2.

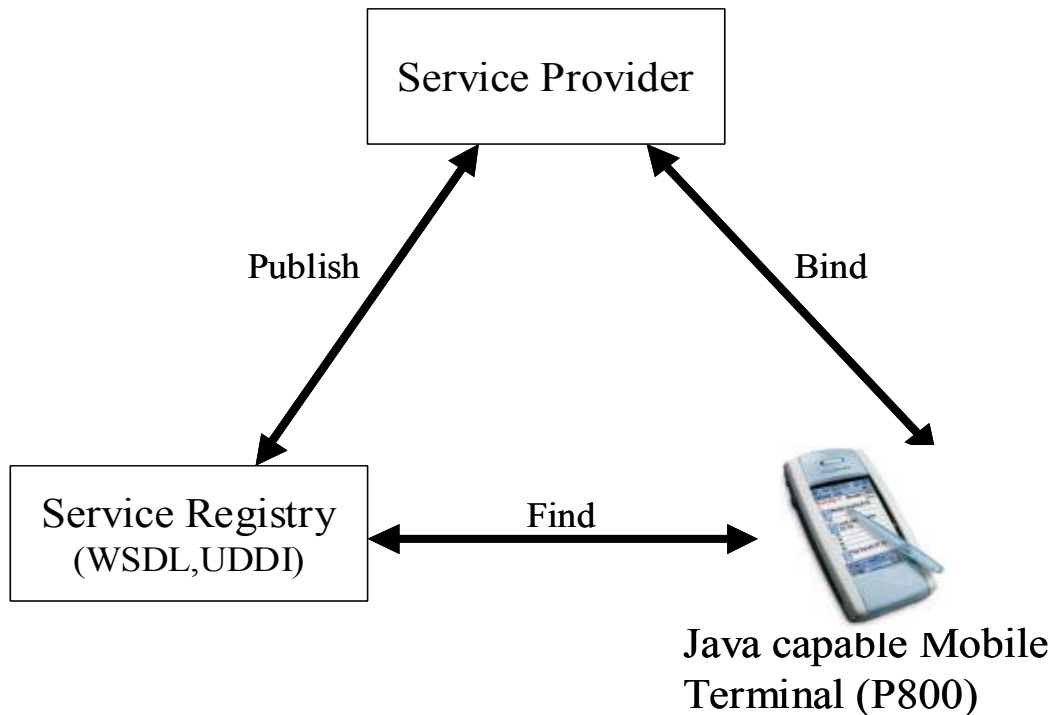


Figure 3.1: Basic architectural setup of mobile Web Service client

The architecture was tested with a WS client implemented on the Sony Ericsson P800² Smart Phone. The implementation and performance analysis details are deferred to chapter 6.1.

¹ Some of these architectures are implemented and verified during the thesis study, and others are left at the conceptual level. The reasons are described in detail at the discussion of these architectures.

² P800 was the Smart Phone from Sony Ericsson, which was used for the implementation of the mobile Web Services. The details of the Smart Phone are available at <http://www.sonyericsson.com/P800/main.htm>

3.2 Mobile terminal as Web Service provider (“Mobile Host”)

3.2.1 Basic architectural setup

Similar to the architecture of mobile terminal as Web Service requestor, the basic architecture of the mobile terminal as Web Service provider can be established as shown in Figure 3.2 with the Web Service provider being implemented on the Smart Phone.

Even though the Web Service provider is implemented on the Smart Phone, the standard WSDL can be used to describe the services, and the standard UDDI registry can be used for publishing and unpublishing the services. This was identified in the study as the Web Service provider was successfully implemented on the mobile terminal with the same general architecture as on any standard desktop system, even under the low-resource considerations of the Smart Phone. The only differences between the ordinary service provider and the Mobile Host¹ would be the number of features supported and the speed of the service provisioning. This has been studied in detail and the results are presented in performance analysis discussed in chapter 6.

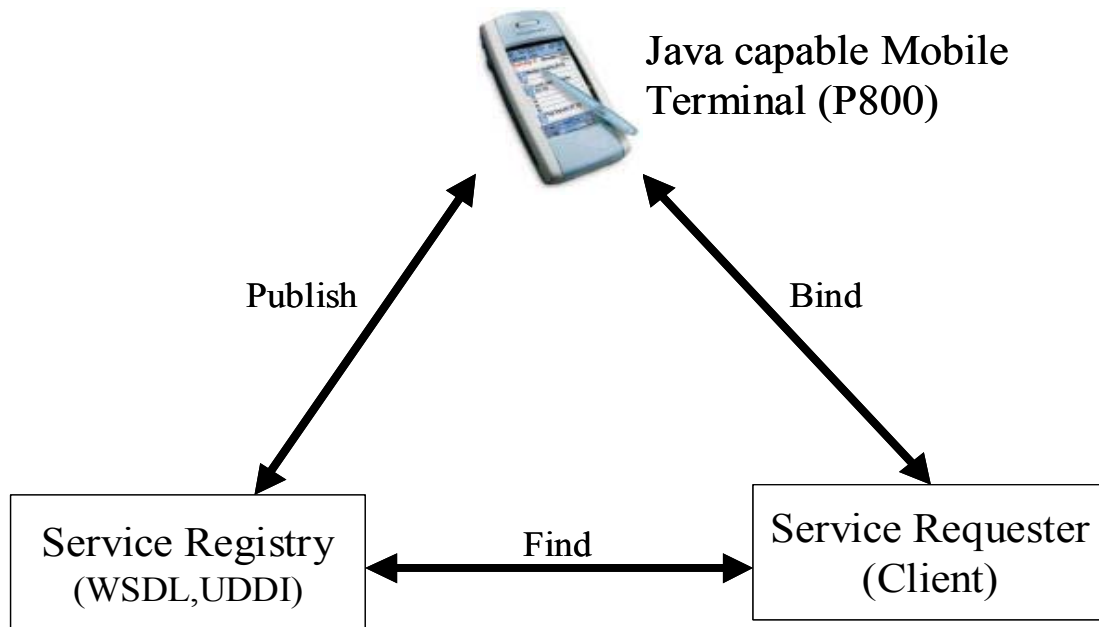


Figure 3.2: Basic architectural setup of Mobile Host

Within the context of the thesis it is always assumed that the client knows the exact location (URI) of the service and the service description. By this the search of the UDDI registry(ies) by the client with the help of the service broker, and the evaluation of any WSDL service description(s), is always bypassed in order to focus more on the main goals of the thesis, i.e. concept, implementation and performance analysis of the mobile Web Service provider for Smart Phones.

¹ From now on the two words “mobile Web Service provider” and “Mobile Host” are used interchangeably through out the document.

3.2.2 Mobile terminal access

Once a Web Service is developed & deployed with the Web Service provider implemented on a mobile terminal (MT), the mobile terminal, that is registered and connected within the mobile operator network, requires some means of identification and addressing, that allows the Web Service to be accessible also from outside the mobile network operator’s network domain.

Generally, computers and devices in a TCP/IP network are identified using an IP address¹. Networks using the TCP/IP protocol route messages (IP envelopes) using their IP destination address. The IP address, that is required for the data transfer to and from Smart Phones (as for any other IP communication client as Web servers, Intranet workstations, etc.) is assigned during the communication configuration phase. Typically, the IP address assigned to mobile devices using GPRS is only temporarily available, and is known only within the mobile operator’s network, which makes it difficult to use the IP number in the client applications.

The following subsections give a detailed description of different possibilities for resolving the IP address and thereby making the data transmission with a mobile terminal, possible.

3.2.2.1 HSCSD

Figure 3.3 below illustrates the architecture that is used to connect the mobile terminal (that provides the Web Service) to the prototyping network using a HSCSD (high-speed circuit switched data) dial-up connection.

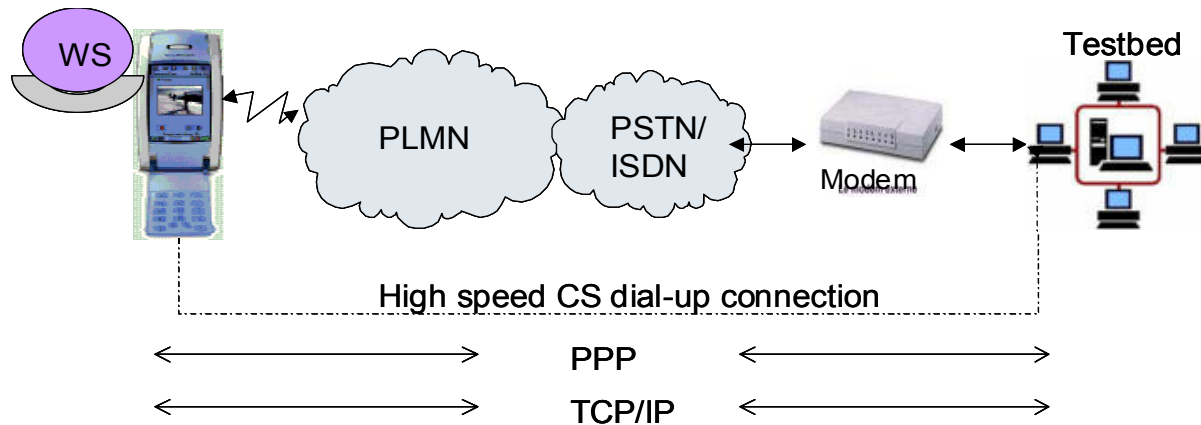


Figure 3.3: Architecture for an end-to-end TCP/IP connection between the mobile terminal and the prototyping network

In this architecture a HSCSD (high-speed circuit switched data) dial-up connection is established between the mobile terminal and an Ericsson prototyping network, the so-called “Testbed”. The connection uses a Public Land Mobile Network (PLMN)² and the Public Switch Telephone Network (PSTN / ISDN) for making the data call to the server. The

¹ More details about IP address at http://www.webopedia.com/TERM/I/IP_address.html

² Details of most of telecommunication acronyms used in this chapter can be obtained at <http://www.webopedia.com>. The detailed description of most of these acronyms is beyond the scope of the document.

3. Architectures

connection is set-up by using PPP (Point-to-Point Protocol) over a circuit-switched data call to a modem that is connected to one of the servers (“Ibiza”) in the Testbed network. On top of this PPP link a TCP/IP end-to-end connection between the mobile terminal and the dial-in server is established. Hence, as long as the data call persists, the mobile terminal can be addressed using the IP address assigned to it by the dial-in server. Thus the Web Service deployed on the mobile terminal can be accessed from any client within the Testbed environment.

The basic concept of the Testbed setup is that all machines are behind a firewall. In addition, this firewall uses NAT (network address translation) in order to map the limited number of public IP addresses to a larger number of internal (private) IP addresses. So, using an appropriate NAT configuration, the mobile Web Service can be accessed by any service requestor (Web Service client) from the Internet. Using the NAT, the Testbed provides a DNS name for the mobile Web Service provider. The only requirement towards the PPP daemon is, that the mobile terminal should always receive the same IP address when it connects to the dial-in server.

This setup is realized and used for the initial testing of mobile Web Services. The connection is a high-speed circuit switched network (28 kbps).

Drawbacks

With the architecture shown in Figure 3.3, the main drawback would be the circuit switched¹ connection, as the connection would have to persist as long as the mobile Web Service provider should be available for the access of its Web Services. In general the billing of circuit switched data connections is based on the time the connection persists, not on the amount of data transmitted across the network. This makes this scenario unfeasible for commercial purposes. Volume based charging is a major advantage enabled by GPRS.

One solution to avoid this drawback is to send an SMS message to the mobile terminal hosting the Web Service provider at the time of a Web Service request towards the mobile terminal, and to have an application running on the mobile terminal, which processes the SMS and starts the connection setup (TCP/IP/PPP over HSCSD dial-up) and Web Service provider. This plan was dropped as the scenario was established for testing purpose only and the method would not be applicable when GPRS connection is established, where it is not problematic to have a continuous virtual connection.

3.2.2.2 GPRS

Once the GPRS connection is established the mobile can be identified by the IP provided by the mobile operator network.

The operational setup for accessing the mobile terminal in a GPRS network is given in Figure 3.4. The mobile TCP/IP connection between the Web Services client and the mobile Web Service provider is deployed on top of a GPRS link into the mobile operator network. From there the traffic is routed through the Internet to/from the Web Service client.

¹ In general you can also have PPP over packet switched connections (which could make sense when you need a end-to-end tunnel through different networks). Usually you use PPP over circuit switched connections as a basis for IP

3.2. Mobile terminal as Web Service provider (“Mobile Host”)

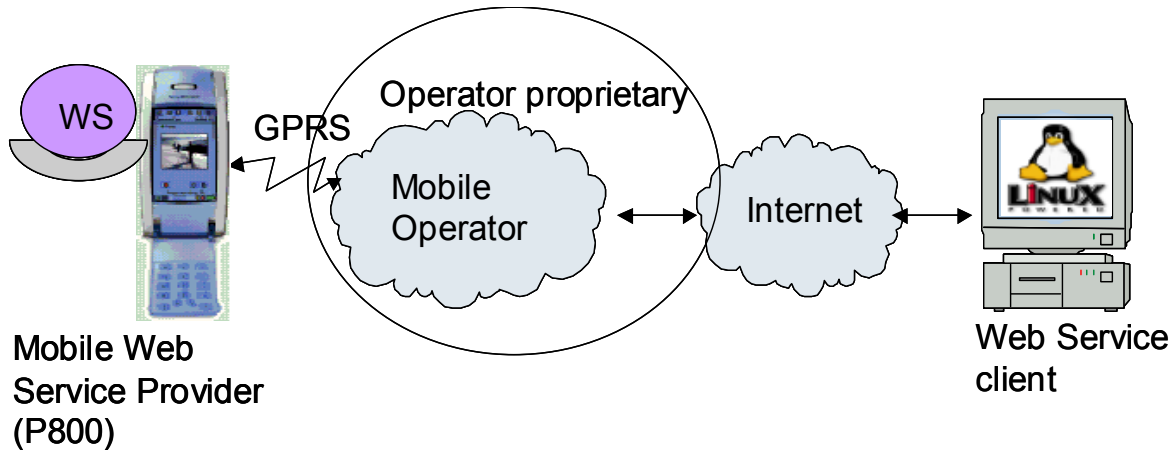


Figure 3.4: The operational setup of Mobile Web Service provider in a live GPRS environment

But there are some serious issues to be considered with such a GPRS connection like: (potentially) limited address range for mobile devices; potential risk of mobile spamming; charging problem of Internet originated / mobile terminated traffic; etc. These are the main reasons for NAT, only private GPRS IP addresses, and no network initiated PDP context activation.

Since the architectural setup shown in Figure 3.4, is not feasible to-date for the study, as the setup needs much support from mobile operator for providing IP, the following architecture shown in Figure 3.5 was used for the basic observation of the Mobile Host scenario in the real GPRS network. The architecture was feasible because of the close collaboration between Ericsson and Vodafone.

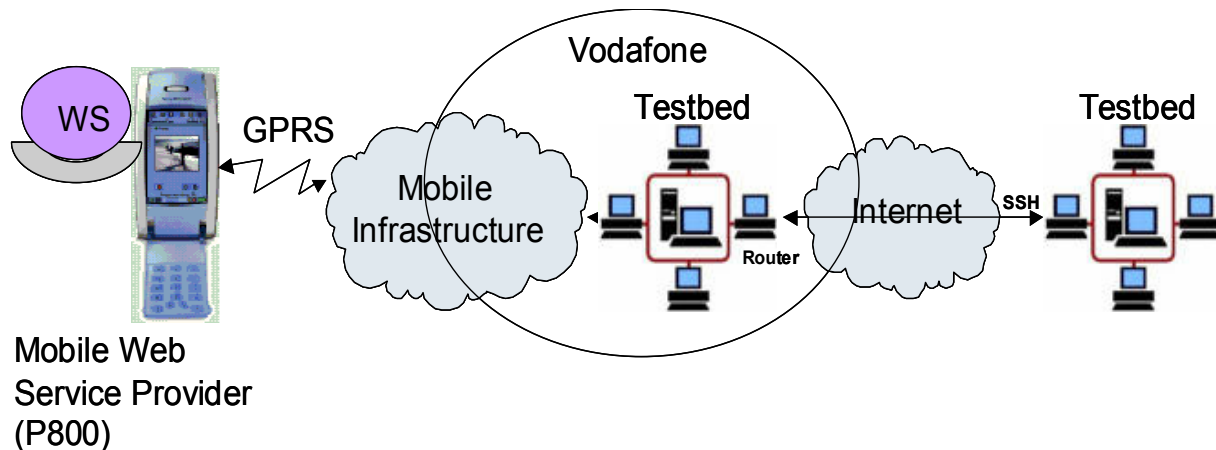


Figure 3.5: The operational setup of Mobile Web Service provider

In this architecture a GPRS connection (PDP context) is initiated at the Mobile Host, and a temporary dynamic IP address is released to the mobile terminal by the mobile operator (Vodafone). The operator also has a Testbed network established, which can be accessed from a system in the Internet using an SSL (Secure Socket Layer) tunnel. An account¹ at the

¹ This access was obtained because of the collaboration of Ericsson and Vodafone.

3. Architectures

Vodafone Testbed then allows to use a SSH (secure shell) on the router in the Vodafone Testbed for the connection with the mobile Web Service provider.

Utilizing the SSL connection, the service requestor can be deployed at the Vodafone Testbed, or can be executed from any computer in the Internet, that supports port forwarding tools like Putty¹.

Port Forwarding is a combination of routing by port combined with packet rewriting. Port Forwarding examines the packet header and forwards it on to another host (after a little header rewriting) depending on the destination port. So the client can be executed on any system and the packets be actually forwarded through the SSL tunnel to the router at Vodafone Testbed.

Drawbacks

The architecture has some serious limitations. It is not feasible for the commercial purposes of third parties², as the architecture directly uses the proprietary network, which may not be suggestible for general purposes.

The IP address provided by the network is temporary, and the WS client must always get the “latest” IP address assigned to the Mobile Host before sending the service request. This makes it feasible only for proprietary applications and for basic testing. This can be eliminated by using NAT at the Vodafone Testbed, providing a DNS name for the mobile Web Service provider.

3.2.3 Alternative scenarios

The subsection gives an overview of the other scenarios and architectures developed and studied during the thesis. First we will discuss a virtual mobile Web Service provider, which was an alternative for the basic architecture for the mobile Web Service provider discussed in chapter 3.2.1. Later we will discuss other potential alternatives for identifying and addressing the mobile Web Services deployed on the mobile terminal.

3.2.3.1 Virtual mobile Web Service provider

During the initial stages of this thesis an architecture for the “mobile terminal acting as Web Service provider” as shown in the Figure 3.6 was discussed.

¹ More about Putty & Port forwarding at <http://the.earth.li/~sgtatham/putty/0.54/html/doc/Chapter4.html#4.19>

² Since the architecture uses the operator’s own network, the operator can use it for his own commercial purposes.

3.2. Mobile terminal as Web Service provider (“Mobile Host”)

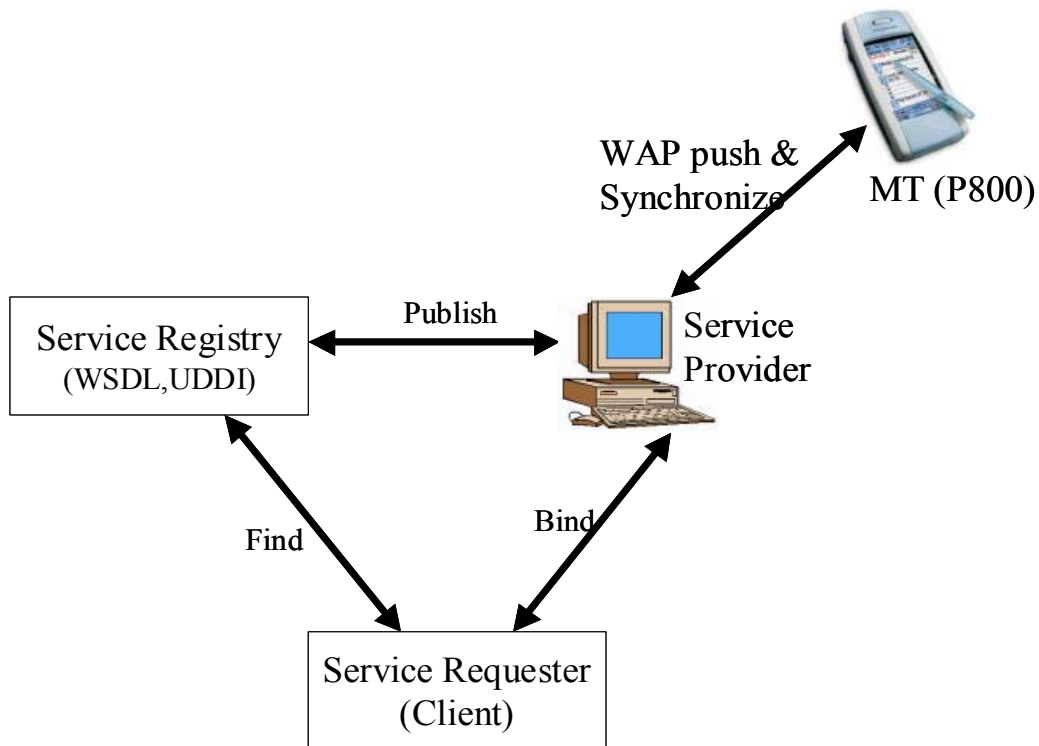


Figure 3.6: The architectural setup of Virtual mobile Web Service provider

In this architecture, two information repositories (like GPS information, pictures, etc.) are maintained, one on the mobile and another on the “real” Web Service provider. These repositories are synchronized regularly using “classic” communication methods, most likely via a GPRS link initiated by the mobile terminal (which “knows” changes of the data, etc.). At the time the “real” Web Service provider receives a request, the service provider uses any means to retrieve the requested information from the mobile terminal, e.g., sending a WAP-Push message to the MT, containing the request and the response URL, pointing to an application on the “real” Web Service provider that expects the information from the MT and relays it to the original external request. By that we would have a (sort of) real-time access to a (virtual) mobile Web Service provider. The communication between the mobile terminal and the real Web Service provider, as explained above, enables the synchronization of the two repositories.

The architecture was planned for implementation as an alternative solution for the case that the Mobile Web Service provider couldn’t be directly developed and implemented on the mobile terminal.

Advantages

With this architecture the basic architectural setup of Web Services is not changed, so this can easily answer many questions, like e.g. How can a (virtual) connection to a mobile terminal that hosts a Web Service, be initiated? How can the mobile Web Service be addressed?, as the answers for general Web Services directly suit for this architecture.

The general UDDI registry can also be used directly, and the critical performance issues like the CPU power/load and memory considerations of the mobile are left for the offline synchronization between the “real” and the “virtual” Web Service provider, and do not influence the real-time performance of a Web Service request.

3. Architectures

However this scenario does not really mean a mobile Web Service provider and therefore was abandoned.

3.2.3.2 Searching for IP at NAT

In the initial stages of the thesis the following architecture was tried to identify and address the mobile terminal in a mobile network. In this scenario the mobile terminal would be connected as a client to a server in the Testbed network via a GPRS link, and it was tried to identify the IP address assigned to the mobile terminal by the GPRS operator. The scenario is given in Figure 3.7.

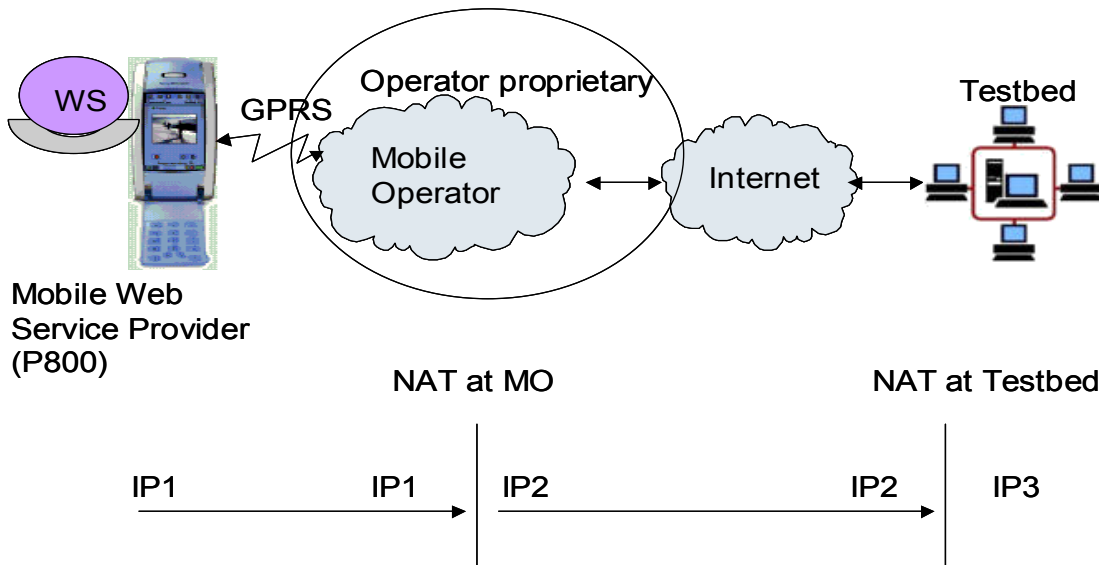


Figure 3.7: The architecture for identifying IP at NAT translation

The tests were conducted on live GPRS network (Vodafone D2 in Germany) and it was assumed that there would be 2 NAT translations during the transmission once at Mobile operator and the other at the Testbed, as shown in the Figure 3.7. Using the IP address in the Testbed (“IP3”), it was tried to reach the IP address of the mobile terminal within the mobile operator network (“IP1”).

Unfortunately the scenario was unsuccessful, as the IP received always was the constant public operator IP address of the NAT proxy (“IP2”) with a varying port number that is mapped temporarily to the mobile terminal connection. This IP address and port number combination can not be used to address the mobile terminal from outside the mobile operator network, as it belongs exclusively to the temporary TCP/IP session and can not be interfered.

But with this architecture it was decided that if the mobile operator provides us with a public IP address, then the setup can be used to connect to the mobile Web Service provider.

3.2.3.3 Workaround with session maintenance

In this architecture we wanted to get the IP address of the mobile by directly getting the details from the mobile terminal by using some standard methods of java like `InetAddress.getLocalHost()` and getting the details by establishing a session with the mobile terminal. The steps in this scenario are described below.

3.2. Mobile terminal as Web Service provider (“Mobile Host”)

- Send request to mobile terminal via WAP-Push (Including server address for response).
- Establishing session on mobile terminal to server
- Respond to server with the IP details.

The scenario is described as in Figure 3.8

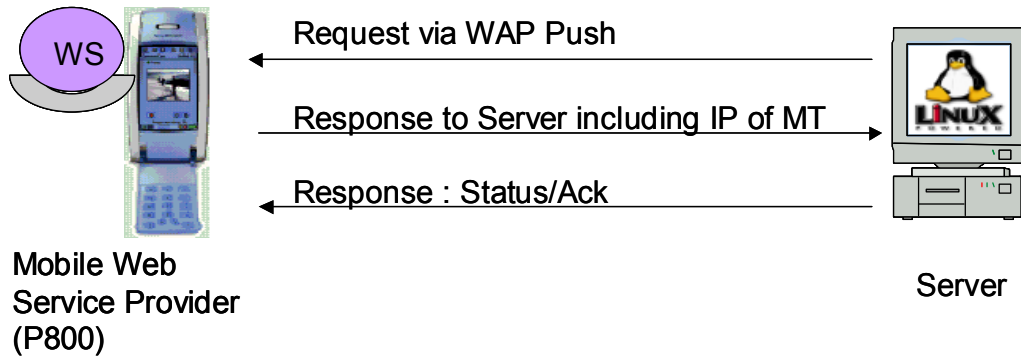


Figure 3.8: The architecture for identifying IP with session maintenance

The scenario was left at the conceptual level itself, and was not implemented, as the thesis had already come up with the solution discussed in the chapter 3.2.2, before this scenario could be implemented. Apart from this, the IP address returned by the MT would have been the same private IP address, discussed in previous subsection (chapter 3.2.3.2) and we would have end up with the same problem (NAT) again.

Summary:

This chapter first described the basic architectures of the mobile Web Services environment, with the mobile terminal as both Web Service provider and Web Service client. Then it described different methods and architectures for identifying and addressing the Web Services deployed on the mobile Web Service provider using the HSCSD and the GPRS connections. It also addressed the feasibility and drawbacks of such connections. Apart from these, the chapter also discussed alternative scenarios considered during the thesis.

3. Architectures

4 Possible applications with mobile Web Service providers

This chapter discusses some of the numerous possible use cases of mobile Web Service providers. The mobile Web Service provider itself gives an opportunity for a new domain of applications, which is yet to be explored in a great deal.

In the following subsections we will discuss some of the applications that have been developed and tested as reference cases for the performance analysis within this thesis. Apart from these, many real-time and business-oriented applications are possible. However, the applications selected here focus on the feasibility¹ for the performance considerations discussed in the following chapters. We also discuss a few more general applications (making use of mobile Web Services technology) in domains like mobile gaming etc.

4.1 Mobile photo album service

Today's high-end mobile terminals as the so-called Smart Phones become more and more advanced, and are generally being equipped with an integrated digital camera. The photographs taken with these mobile phones can later be uploaded or transferred to PCs through cables or by using wireless methods like Infrared or Bluetooth.

Using currently available technologies, if a user wants to publish the photographs he had taken with the mobile terminal to the public or friends, he has to upload the photos to a Web server, from which they can be accessed. The user can also send the images through MMS or some other means of messaging to the clients. Here the mobile owner bears the payment for the communication between his Smart Phone and the Web server or the receiver's device.

With a mobile Web Service provider, implemented and deployed on the Smart Phone, interested people can access the Mobile Host using a standard Web Service client or a Web client, and can browse through the pictures they are interested in. This is comparable to any other online image album service, but implemented on the mobile terminal.

Applications developed for a Mobile Host also have to consider security and authorization details. The applications can be upgraded to take the manual permission of users, before providing the pictures (services) to the client.

This mobile host application was implemented and developed using PersonalJava, and has been deployed on a SonyEricsson P800 Smart Phone, for the analysis of the performance characteristics of a this type of a Mobile Host.

It provides support for research applications like biological research, news/sports etc., where the results of experiments can be taken directly and can be provided automatically for those who are registered for that service and interested in the details. The scenario of the application is clearly shown in the Figure 4.1.

¹ We will discuss these details in greater detail in the subsequent subsections of this chapter

4. Possible applications with mobile Web Service providers

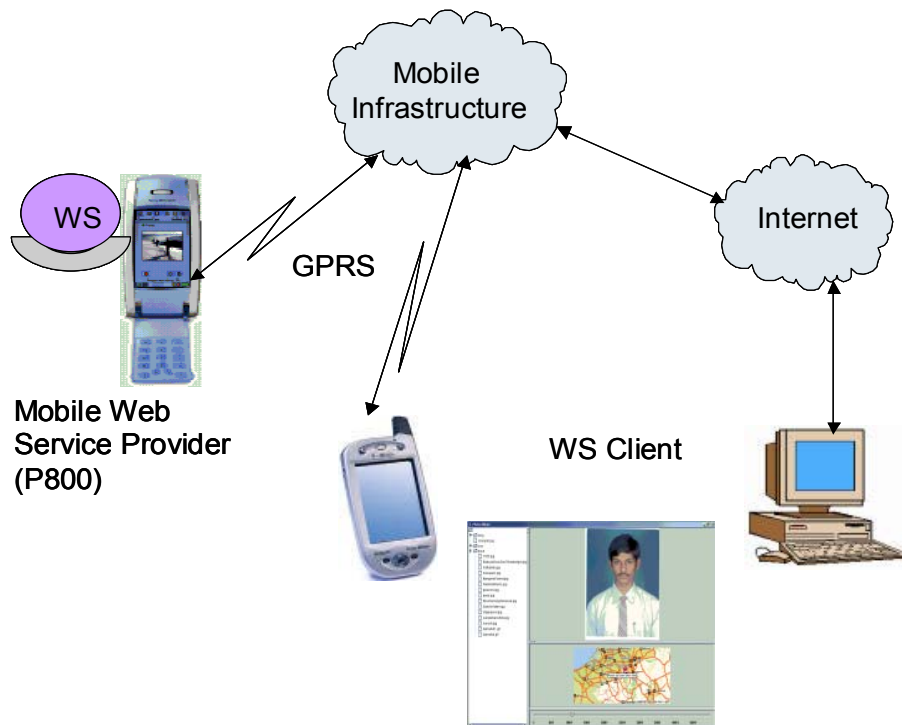


Figure 4.1: The mobile photo album service scenario

The application was considered for implementation because of the greater scope for observation of different parameters at performance analysis like the serialization overhead, deserialization overhead, and transmission delay etc. apart from its real time use. The parameters of consideration and the application are discussed in more detail in the performance analysis in chapter 6. The implementation details are discussed in detail in next chapter.

4.2 Location (GPS) data provisioning

The Mobile Host can also be used for providing detailed location information of the mobile terminal, such as GPS¹ (Global Positioning System) data², in conjunction with a corresponding module or external devices. A dedicated Web Service provides these GPS data upon request through the mobile Web Service provider. The GPS data can also be stored in the Mobile Host at a particular instance, for example when taking a picture, and can later be provided as additional GPS information together with the picture. The application shown in Figure 4.1 uses both services explained above, and provides pictures from the mobile album, as well as location details of the pictures.

The above two applications discussed, as a combination could also be of use, for example in a distress call, the mobile terminal could provide a geographical description of its location along with location details. An interesting example of the application explained above would be a use case for editors and journalists. The Mobile Host can be used in this scenario for the co-ordination between journalists, who are covering different events across globe, and their respective organizations. The scenario is illustrated in Figure 4.2.

¹ More details about GPS can be found at <http://www.trimble.com/gps/what.html>

² GPS data format is available at <http://www.navcen.uscg.gov/pubs/gps/sigspec/gpssps1.pdf>

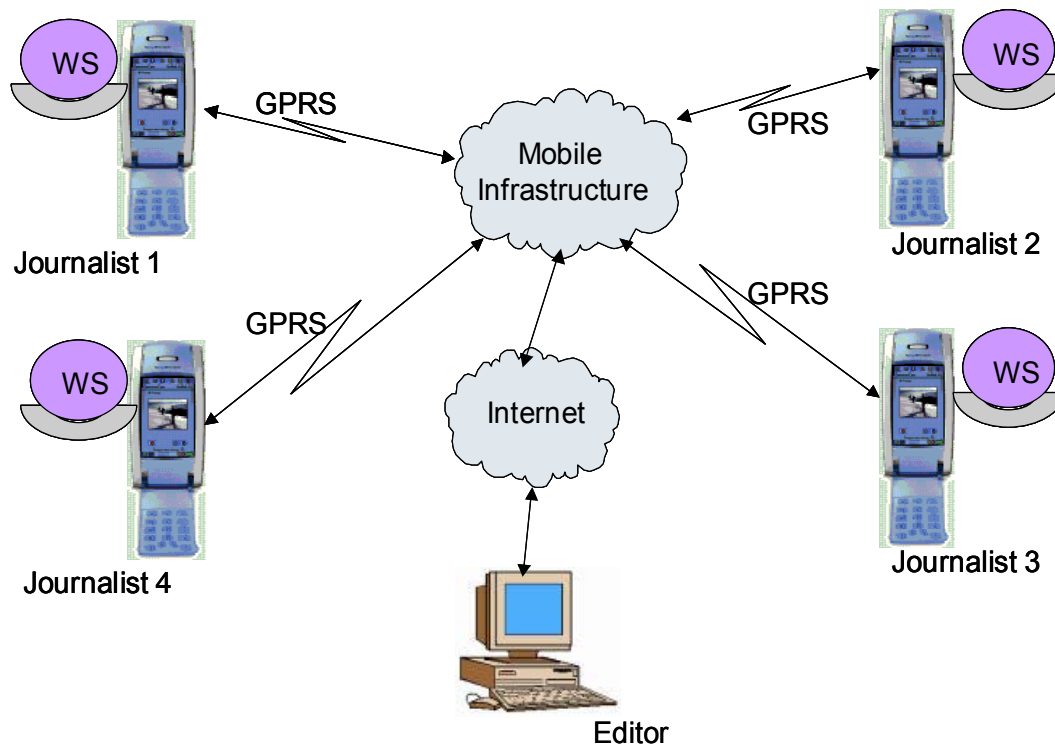


Figure 4.2: The Mobile Host used in Journalism scenario

In the scenario shown in Figure 4.2, journalists can be at different locations across the globe, covering different events like the sport events, conferences, etc. An editor can always keep track of the location of “his” journalists and the content they have gathered. He can browse through the pictures taken by the journalist at any instance. Standard client applications can be developed for the editor, which synchronize the information stored by editor and data at the Mobile Host. By such an application the journalists can concentrate more on their job of collecting, as they don’t have to upload the data every time they get something interesting.

4.3 Mobile gaming

Today, there are high expectations and hopes towards mobile gaming. Mobile terminals become more and more powerful, multi-media capabilities are constantly extended, and the evolution of the mobile networks enables more and more interactive mobile applications. Hence, many mobile operators are seeing them as one of the next generation business drivers in the mobile arena.

In mobile gaming, game clients are connected to a Game Access Server (GAS)¹ via an access system such as wireless networks. The client software is installed on the mobile terminals that interface the game application on the local GAS. Game Access Servers are situated at the edge of data networks, and are operated by telecom operators, ISPs, or game service providers. Each GAS holds a copy of the game state and distributes game data to clients according to priorities that are set by the clients. Game clients that belong to different access units may participate in the same game. Game Access Servers send aggregated game data streams to each other over backbone networks.[20]

¹ More details about multi-user gaming and GAS at http://www.wirelessdevnet.com/symbian/rb_20.html

4. Possible applications with mobile Web Service providers

The Mobile Host could be very useful in the mobile gaming domain. Until now, as explained earlier, a GAS is always involved in the coordination between the players, providing the ad-hoc characteristics to the network. Now, with the Mobile Host in the scenario, the GAS functionality can be implemented by one of the players and can act as GAS, providing games and coordinating the communication between the players, thereby creating scope for a total new set of games and payment methodologies, which can involve third party developers and individuals as game organizers, supporting ad-hoc networks. The scenario is shown in Figure 4.3.

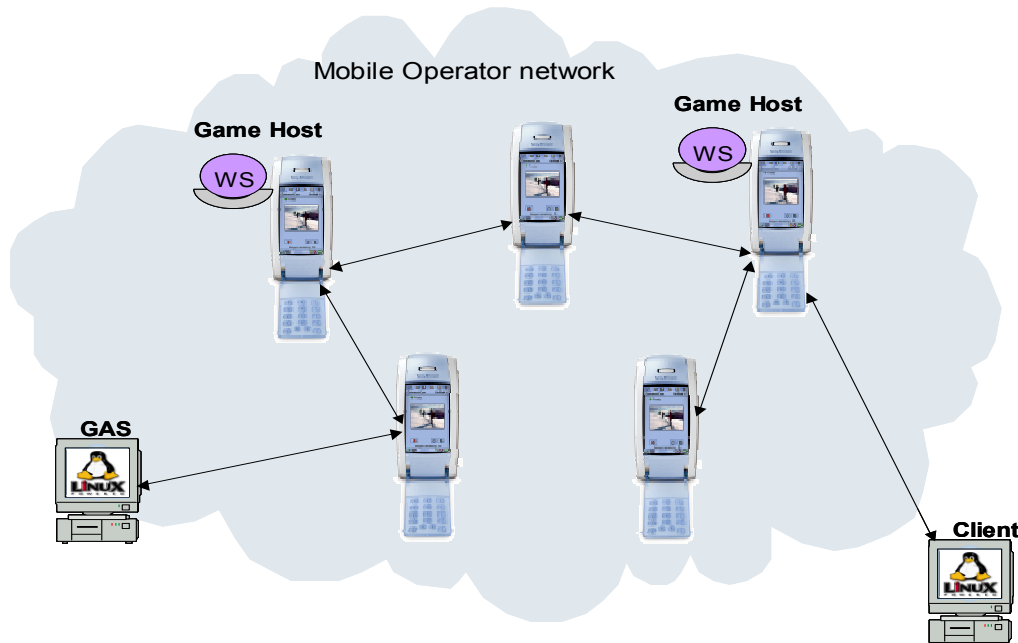


Figure 4.3: Scenario with Mobile Host in ad-hoc mobile gaming environment

The Mobile Host can act as GAS for other mobile phones in the mobile operator network, or for external clients connected to it via Internet. The game played by the mobile client, can also span across multiple such Mobile Hosts, which co-ordinate the game provisioning, thereby making an ad-hoc gaming network based on mobile phones.

Still, there are many open questions to be studied in this area, as the Mobile Host itself gives scope for new distributed architectures for wireless networks, and thus can support new payment methodologies, communication and co-ordination of different participants of different games etc.

4.4 Transportation and Logistics

The Mobile Host offers further deployment opportunities in the transportation and logistics domain. Example areas are maintaining traffic control¹, and guided parcel service etc. The scenarios are explained briefly below.

Most of today's traffic congestions are caused by excessive traffic or by construction sites. Nowadays, scientists studying traffic congestion and its causes, have a pretty clear picture of how a small incident or disturbance can develop into a full-fledged traffic jam and the means

¹ More details available at <http://www.invent-online.de/en/projects.html>

of avoiding such a situation. In the future, driving assistance systems will utilize these scientific results, collecting and combining data dynamically to analyze and even predict the traffic state traffic volume, speed, density, and to modify speeds, headways, and lane changing so as to keep traffic flowing as smoothly as possible even under adverse conditions. By making traffic itself behave intelligently in this way, traffic performance assistance systems will help to minimize spontaneous disturbances that otherwise could cause congestions or stop-and-go traffic.

The driving assistance systems introduced above can be provided with Mobile Hosts, which allows them to respond according to the incoming WS requests and take alternative measures. The Mobile Host can also help in the inter-vehicle communication, which warns the subsequent vehicles. So if a traffic jam is already present, traffic performance assistance systems can be designed to minimize the length of the jam and to adapt vehicle speeds so as to return faster to un-congested traffic. The application scenario is shown in Figure 4.4.

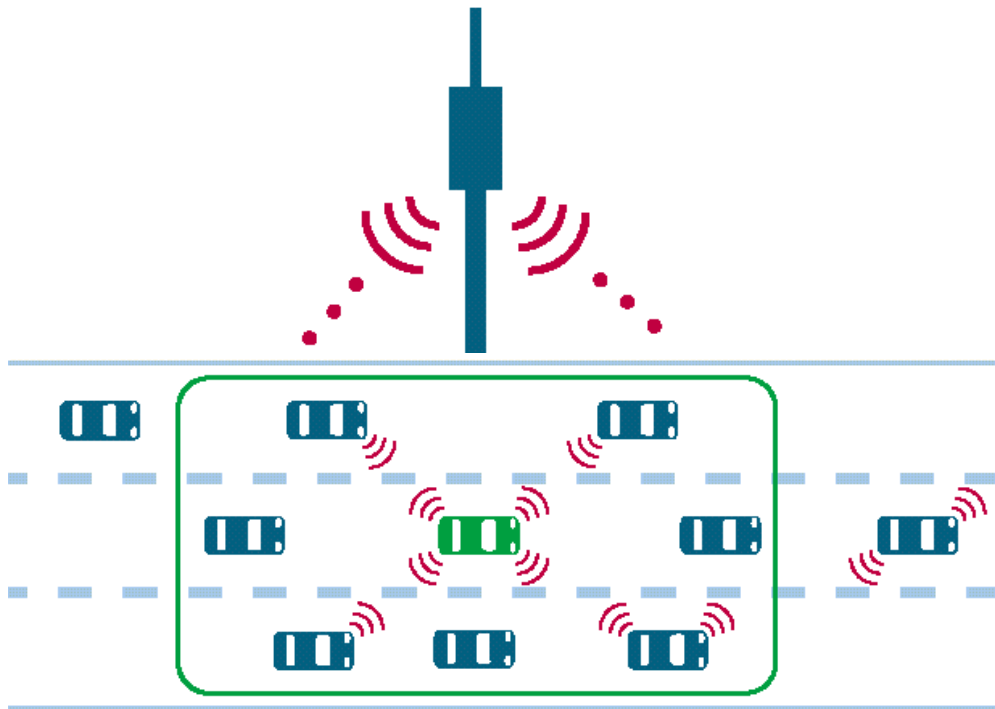


Figure 4.4: Mobile Host used in traffic control systems

The figure shows the communication of driving assistance systems, provided with the vehicles, with external towers. It also shows inter-vehicular communication. Using such applications, existing roadway networks could be utilized much more efficiently to meet traffic demand and avoid overloads.

In similar lines to the above described traffic control maintenance, proprietary applications can be developed on the Mobile Hosts, which allow the vehicle drivers to receive requests from different clients for the parcel delivery. Such a system can also answer client requests like pickup location, change in delivery address, duration for delivery, and the destination of the parcel etc. which also allows the client properly co-ordinate with the parcel delivery system. Several of these parcel delivery vehicles can be coordinated by a common server application, which identifies the most feasible vehicle based on the client request details,

4. Possible applications with mobile Web Service providers

and forwards the request from the client to that vehicle, based on some parameters like the distance from the pickup location, the delivery location etc. The application can also benefit from the GPS service discussed earlier. The scenario is shown in Figure 4.5.

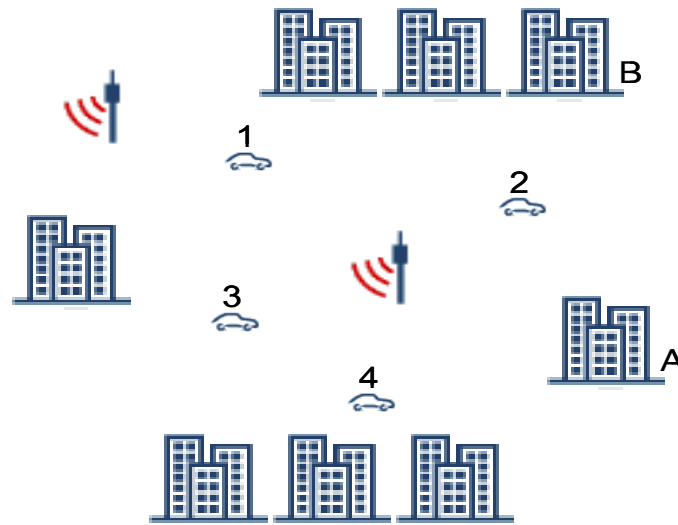


Figure 4.5: Guided parcel service scenario

All the vehicles shown (1,2,3,4) are provided with Mobile Hosts. A client can contact the vehicle directly, or through the central system, shown by towers, and ask for a service that takes parcel from location A and delivers it at location B. If the client calls the central system, the system diverts the request to the best feasible vehicle that can serve the client. If the client calls the vehicle directly, the vehicle can co-ordinate with other vehicles and the central system to find the most feasible vehicle. The best advantage of such a system is, that, the client can exactly track his parcel throughout the delivery process.

Apart from the applications described here, a Mobile Host can also be used in many other scenarios like home equipment control, robot control, e-medicine etc. because of its low resource requirements. In all these applications the equipment or the appliances are connected to the mobile Web Service provider by some means like Bluetooth, Infrared and etc., through which the Mobile Host can deliver respective commands to the devices based on the WS requests it receives.

The following Table 4.1 summarizes different use cases discussed in this chapter. The table mainly considers the amount of communication and co-operation needed for the establishment of connections (Ad-hoc or Peer-peer) for the respective applications. The interaction between the Mobile Host and the external media is also analyzed in the table.

Scenario	Communication	Cooperation	Data provision	Human to machine interaction	Machine to machine interaction
Mobile photo album service	X	-	X	X	-

Location data provisioning	X	-	X	X	-
Mobile gaming	Supports ad-hoc communication	X	X	X	X
Transportation and logistics	Supports ad-hoc communication	X	X	X	X
Home equipment control	X	X	-	To some extent	X
Robot control	X	X	-	-	X
e-medicine	X	-	X	X	-

Table 4.1: Summary of different possible use cases of Mobile Host

Summary:

The chapter discussed some of the applications that have been developed and tested as reference cases for the performance analysis within this thesis. The chapter also discussed some domains like mobile gaming, transportation and logistics, where the Mobile Host can be incorporated.

4. Possible applications with mobile Web Service providers

5 “SSNServer” – Mobile Web Service Provider

As stated earlier, the main objective of the thesis is to study the concept, implementation and performance testing of mobile Web Service provider for Smart Phones. So for this purpose a Mobile Host (SSNServer) was developed for the limited-resource devices, especially Smart Phones. The chapter discusses the technologies supported by these low-resource devices, the HTTP protocol and the messages and then the architectural and implementation details of the Mobile Host.

5.1 Types of java for mobile phones

Since the main interest of the thesis is to study the concept, implementation and performance of Mobile Host, the Java version, used for the implementation of the mobile Web Service provider, is selected under consideration of specific criteria;

- The processing power and speed of the mobile terminal.
- Extended support of the Java version by the SonyEricsson P800 Smart Phone, which was selected as terminal for the implementation and performance analysis.
- Portability of the application.

The types of Java applications that can be developed for the Smart Phones depend on the Java support in these terminals. The SonyEricsson P800 Smart Phone supports two types of Java, namely PersonalJava and J2ME CLDC/MIDP 1.0.

The following subsections discuss both of the technologies mentioned above.

5.1.1 PersonalJava

PersonalJava¹ also referred as ‘pJava’, was one of the first Java programming environments targeted at developing applications for resource-constrained devices. These devices include Smart Phones, PDAs (Personal Digital Assistants) and many other embedded devices. PersonalJava specifies a reduced set of class libraries compared to the desktop environment. Over the years, the Symbian port of PersonalJava to SymbianOS² has been undergoing substantial optimizations like assembler coded byte code interpreter, optimized AWT³ library etc., that today PersonalJava in combination with the hardware performance enhancements is a powerful alternative for the development of mobile applications.

The PersonalJava profile is based on the JDK1.1⁴ (Java Development Kit), but makes a number of packages, classes, and methods optional. It gives the capability to the developer to create Web applets and other mobile phone applications. It is the first attempt by Sun to produce a Java application environment (JVM) for mobile devices; it predates the Connected Device Configuration (CDC) and is the forerunner to J2ME.

¹ More details about PersonalJava can found at <http://java.sun.com/products/personaljava/>

² More details about SymbianOS at <http://www.symbian.com/>

³ AWT (Abstract Window Toolkit) is a part of the Java Foundation Classes (JFC) -- the standard API for providing graphical user interfaces (GUIs) for Java programs.

⁴ More details about JDK 1.1 are available at <http://java.sun.com/products/archive/jdk/1.1/index.html>

5.1.2 J2ME – Java 2 Platform, Micro Edition

Java 2 Platform, Micro Edition (J2ME)¹ is a subset of the Java 2 Platform, Standard Edition (J2SE)², and is the Java platform for consumer and embedded devices such as mobile phones, PDAs, TV set-top boxes, in-vehicle telematics systems, and a broad range of other embedded devices. J2ME is defined through the Java Community Process, and it also maintains the Java philosophy of portability.

Sun has divided the Java 2 platform into different editions based on their end system requirements. The different platforms and their areas of focus are illustrated in Figure 5.1.

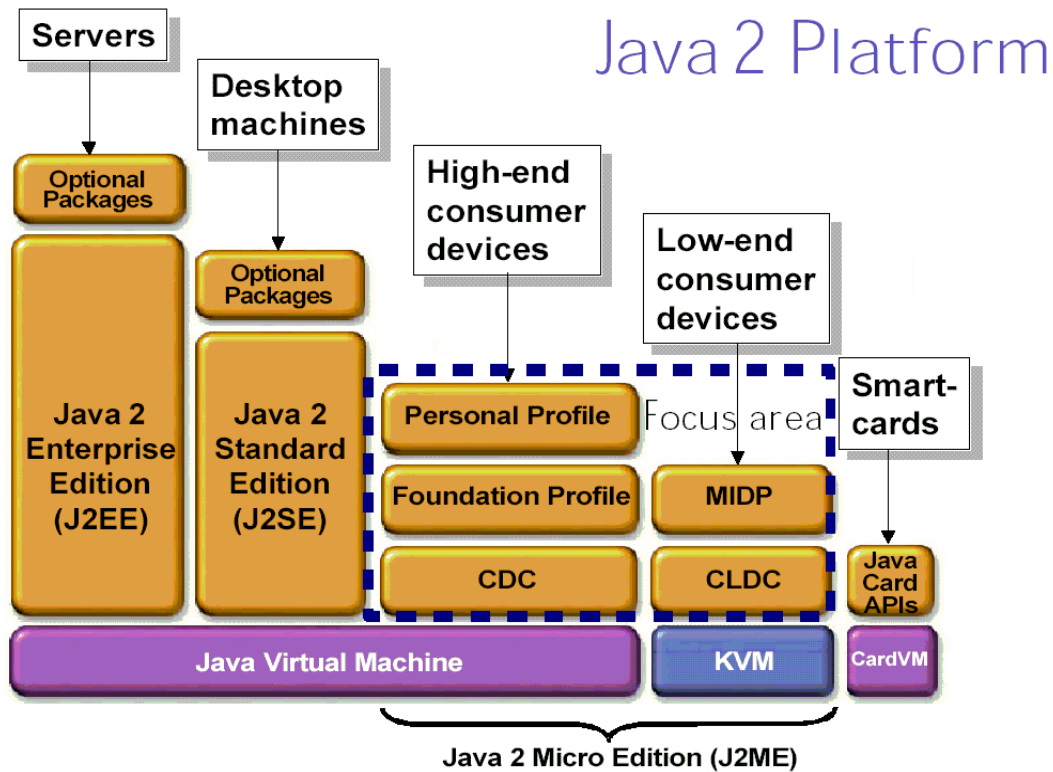


Figure 5.1: Architecture of Java 2 Platform [7]

J2SE provides a complete environment for application development on desktops and servers. It also serves as the foundation for the Java 2 Platform, Enterprise Edition (J2EE)³, which is used for developing multi-tier enterprise applications.

J2ME was designed by grouping devices into “configurations”, a vertical set of virtual machines and minimal set of class libraries providing the base set of functionality for the range of devices within each configuration.

¹ Sun provides a detailed discussion of J2ME at <http://java.sun.com/products/archive/jdk/1.1/index.html>

² More details at <http://java.sun.com/j2se/index.jsp>

³ J2EE specification is available at <http://java.sun.com/j2ee/index.jsp>

Two configurations have been defined for J2ME, the Connected Limited Device Configuration (CLDC)¹ and the Connected Device Configuration (CDC)². The CLDC is the smaller of the two platforms, and caters for devices such as mobile phones with an intermittent network connection, slow processor (16 or 32 bit), and limited memory (128kb-512kb). The CDC is designed for devices such as high end PDAs and communicators, with more memory (minimum of 2Mb) and a 32-bit processor.

The development environment for J2ME on the CLDC devices is the Mobile Information Device Profile (MIDP)³. It defines the classes for user interface, persistent storage, and networking and application management. Combined with CLDC and its Kilo Virtual Machine (KVM) [8], this profile provides a complete Java runtime environment (JRE)⁴ for handheld devices with minimum memory and processor power. It allows programs to be downloaded, over the air to the device from a service provider, in the form of MIDlets⁵.

The Personal Profile is the CDC profile, aimed at devices that require full Graphical User Interface (GUI) or Internet applet support, such as high-end PDAs, communicator -type devices, and game consoles. It includes the full Java Abstract Window Toolkit (AWT) libraries and offers Web fidelity, easily running Web-based applets designed for use in the desktop environment. Personal Profile replaces PersonalJava technology, and provides PersonalJava applications a clear migration path to the J2ME platform.

Combining various optional packages can further extend the J2ME platform. These optional packages along with CLDC, CDC, and their corresponding profiles, can address very specific market requirements. These optional packages offer standard APIs to support both existing and emerging technologies like Bluetooth, Web Services, wireless messaging, multimedia, and database connectivity etc.

After the analysis of both the development platforms, PersonalJava was selected for the implementation of the Mobile Host on P800, since:

- PersonalJava has a richer application environment and can interact more extensively with the phone software than J2ME.
- PersonalJava is faster in processing on the P800 Smart Phone [12].

Using PersonalJava may affect the portability of the Mobile Host on different Smart Phones, but this is not a big constraint for this thesis, as the main interest is on the concept and performance analysis of the mobile Web Service provider on the given Smart Phone.

5.2 HTTP (Hypertext Transfer Protocol)

The Mobile Host is developed as a Web Service handler, built on top of a normal HTTP Web server. The architecture and the implementation details of the Mobile Host are explained in detail in the later parts of this chapter.

¹ More details about CLDC at <http://java.sun.com/products/cldc/index.jsp>

² More details about CDC at <http://java.sun.com/products/cdc/index.jsp>

³ For further information on MIDP refer to <http://java.sun.com/products/midp/index.jsp>

⁴ JRE allows end-users to run Java applications.

⁵ More details about MIDlets at <http://developers.sun.com/techtopics/mobility/midp/questions/spawn/>

5. “SSNServer” – Mobile Web Service Provider

The Web Service requests are passed to the Mobile Host by HTTP tunneling¹. Therefore, before the detailed description of the architecture of the Mobile Host, we will discuss briefly HTTP² (Hypertext Transfer Protocol), HTTP messages, and the Web server basics, to the extent required for the further discussion of the architectural details of the Mobile Host.

HTTP is the application layer protocol of the Web. It is implemented in two communication peers: the client program and the server program, executed on different end systems, and communicating through HTTP messages. HTTP defines the method and the structure of message for the communication.

HTTP defines how the Web clients (browsers) request Web pages from the Web servers and how the Web servers transfer the Web pages. The HTTP client first initiates a Transmission Control Protocol³ (TCP) connection to the server port (default port for HTTP is port 80). Once the TCP connection is established, the client sends the HTTP request message to the server through the socket associated with the TCP connection. The message includes the path name⁴ of the Web page. The HTTP server receives the request through the socket, and retrieves the Web page from the storage (file system or RAM) of the server, encapsulates the object in an HTTP response message, and sends it to the client using the socket. Once the client receives the message, the TCP connection is closed.

The HTTP as explained above uses the non-persistent TCP connection mode, as the TCP connection closes after processing the client request. A persistent TCP connection is also possible, where the server maintains the connection even after sending the response. Subsequent communication between the same client and server utilizes the already established and still open connection.

5.2.1 HTTP message format

The HTTP specification defines two types of message formats, HTTP request and HTTP response messages. The following subsection gives a brief description of these message formats. A detailed description of the messages is available with the HTTP specification (RFC2616).

5.2.1.1 HTTP Request message

The HTTP request message is just a normal ASCII text message. The client formulates this message, and sends it to the Web server for processing. The message format is shown in Figure 5.2. A brief description of the fields of the HTTP request message, that are relevant for the thesis, is given below.

¹ Using Http tunneling it is possible to send data of any protocol through proxy over HTTP. The protocol messages are wrapped into the Http message body and are transferred as normal Http GET/POST requests.

² Detailed specification of HTTP is available at <http://www.w3.org/Protocols/>

³ More details about TCP at http://vit.smolensk.ru/docs/network/illustrated_tcp_ip/index.html

⁴ The HTTP message formats are discussed in detail in the next subsection

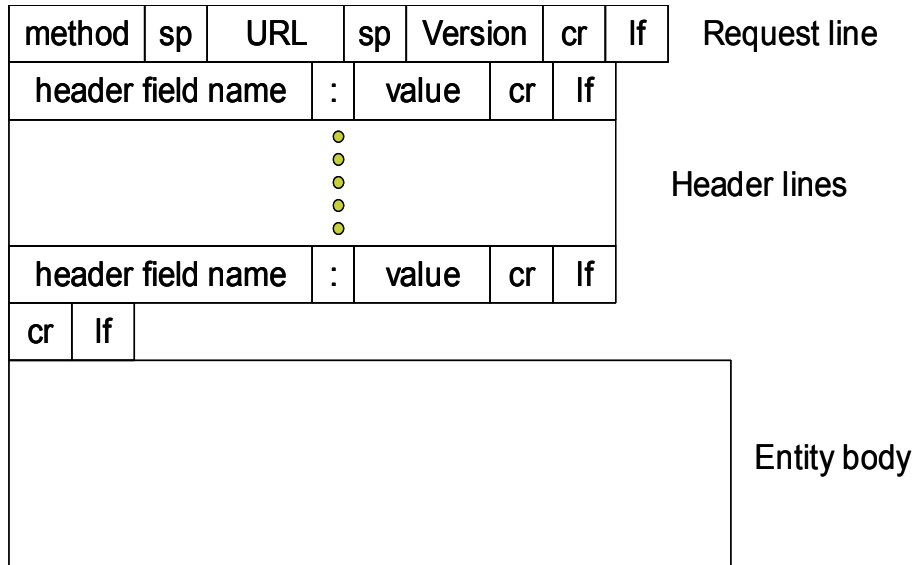


Figure 5.2: General format of HTTP request message

Method

The Method field indicates the method to be performed on the object identified by the URL. Methods can be GET, which means retrieval of the data the URL identifies, HEAD, which is the same as GET, but returns only the HTTP headers and no document body, and POST, which creates a new object linked to the specified object. The message-id field of the new object may be set by the client or else will be given by the server. A URL will be allocated by the server and is returned to the client. The new document is the data part of the request

Version

The Protocol Version field defines the format of the rest of the request. If the protocol version is not specified, the server assumes that the browser uses HTTP version 1.0.

Entity body

The content of an object is sent (depending on the method) with the request. The entity body is used with POST method and not with GET method.

5.2.1.2 HTTP Response message

Similar to request message the HTTP response message is also written in normal ASCII text. The response message format is shown in Figure 5.3. A brief description of the fields of the HTTP response message, that are relevant for the thesis, is given below.

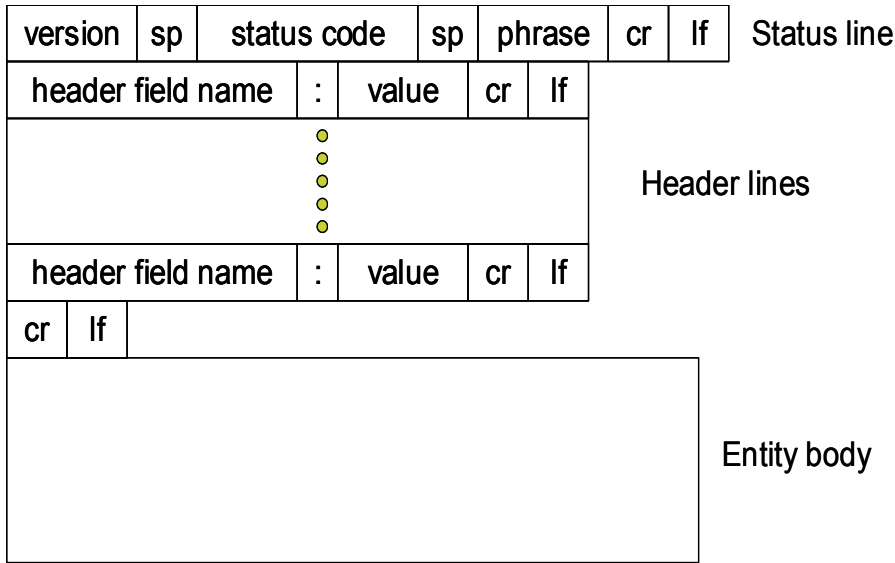


Figure 5.3: General format of HTTP response message

Version

The version field specifies the HTTP version used by the server.

Status code

The status code and the phrase associated with it, indicate the result of the request. For example, “200 OK” indicates the request is successful and the information is returned in the response, “404 Not Found” indicates that the requested document does not exist on the server.

Entity body

The entity body is the main part of the message; it contains the requested object itself.

5.3 Architecture and features of the Mobile Host

This subsection discusses architectural details of the mobile Web Service provider implemented and used for the performance analysis.

The Mobile Host has been developed as a Web Service handler built on top of a normal Web server. The normal HTTP (Hypertext Transfer Protocol) requests are processed as generic HTTP Web server requests, and the Web Service requests sent by HTTP tunneling are diverted and handled by the Web Service handler.

Figure 5.4 shows the core architecture of the Mobile Host. At the HTTP interface a server socket is waiting for the incoming HTTP GET/POST requests. When the Mobile Host gets a HTTP request, the server socket accepts the request, creates a socket for communication, and initiates a new thread of execution by creating an instance of the request handler. The request handler extracts the incoming request data from the input stream of the socket, and checks for the “SOAPAction” header field. If the header value is not set, the request handler processes the HTTP request just as the normal Web server (HTTP server), and returns the response by writing to the output stream of the socket.

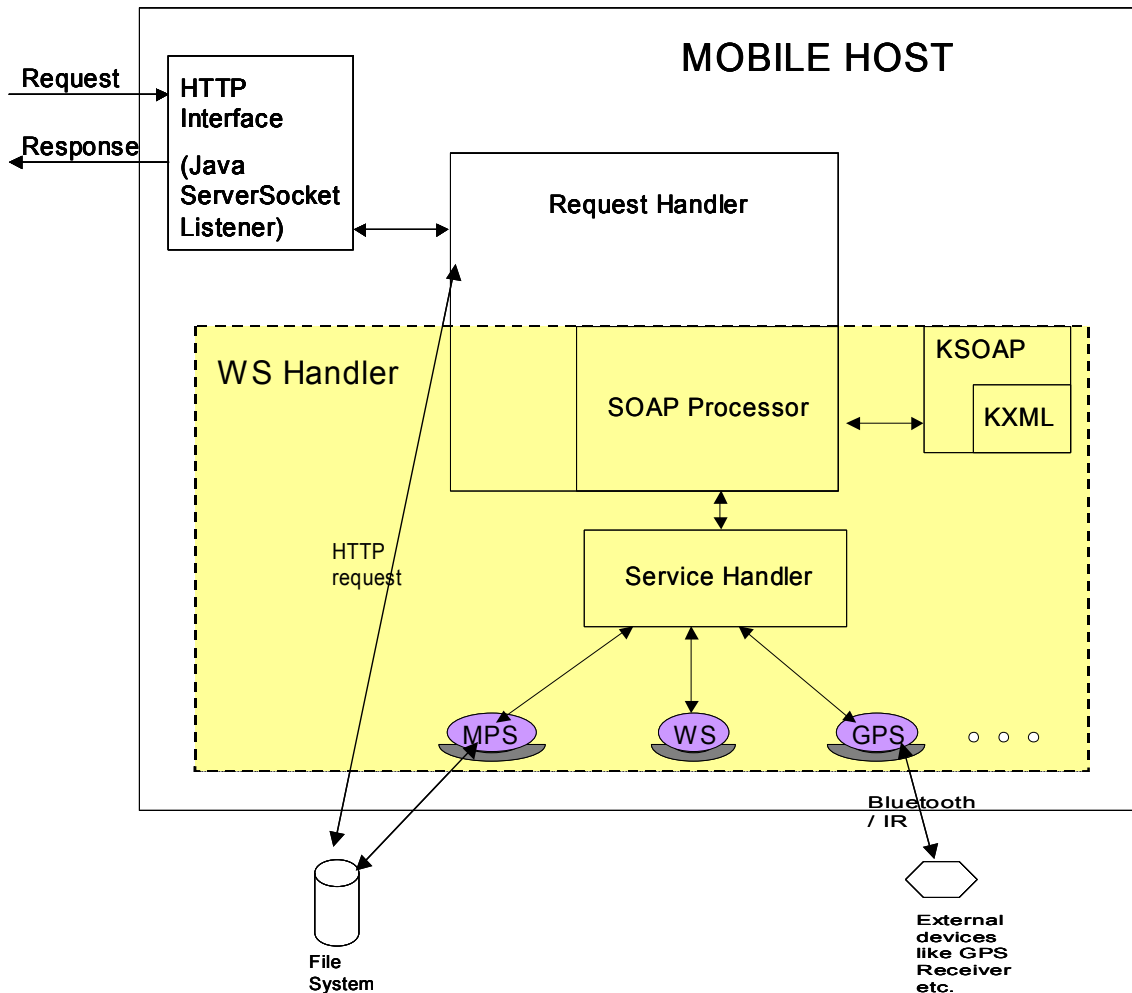


Figure 5.4: Core architecture of the Mobile Host

If the “SOAPAction” header value is set, then the request handler reads the HTTP message body and deserializes¹ the SOAP request (XML data structures) to Java objects using the kSOAP and kXML. The request handler passes these objects to the service handler, which extracts the request parameters and calls the respective services including the parameters. The business logic of the service method is then executed and the service handler returns the response to the request handler.

The Web Services deployed on the Mobile Host can access the local file system, or any external devices like a GPS receiver, using Infrared, Bluetooth etc., and can implement business logic.

The request handler serializes the response and prepares the HTTP response message, which is returned to the client as a HTTP response by writing to the output stream of the socket.

The activity diagram shown in Figure 5.5 explains the process in detail. The swimlanes indicate the node carrying out the functionality.

¹ The terms deserialize and serialize are briefly described after the architectural details of Mobile Host.

5. “SSNServer” – Mobile Web Service Provider

Before considering the implementation details, we will discuss some of the concepts mentioned here in the architecture.

Deserialization:

As discussed earlier¹, SOAP messages are in the form of XML data structures. So, to extract the details of the requested services (like the service name, input parameters for the service etc.), that are incorporated in the ASCII based XML data structures, the XML data is to be parsed and the details are to be returned as objects, so that the service details can be extracted as and when necessary. This process is called deserialization. The process returns “SOAP Envelope” objects. Different implementations of SOAP have their own support for deserialization. We had already discussed kSOAP’s support for serialization and deserialization.

Serialization:

When the response of the Web Service is to be sent back to the client as SOAP response messages, the SOAP response message objects on the Mobile Host are to be converted back to the XML messages streams. This process is called serialization, which produces an XML stream that can be transferred across the standard IO streams as normal ASCII stream, and works analog to deserialization.

¹ During the SOAP and kSOAP discussion in chapter 2, we had considered the structure of SOAP messages and the process of serialization and deserialization

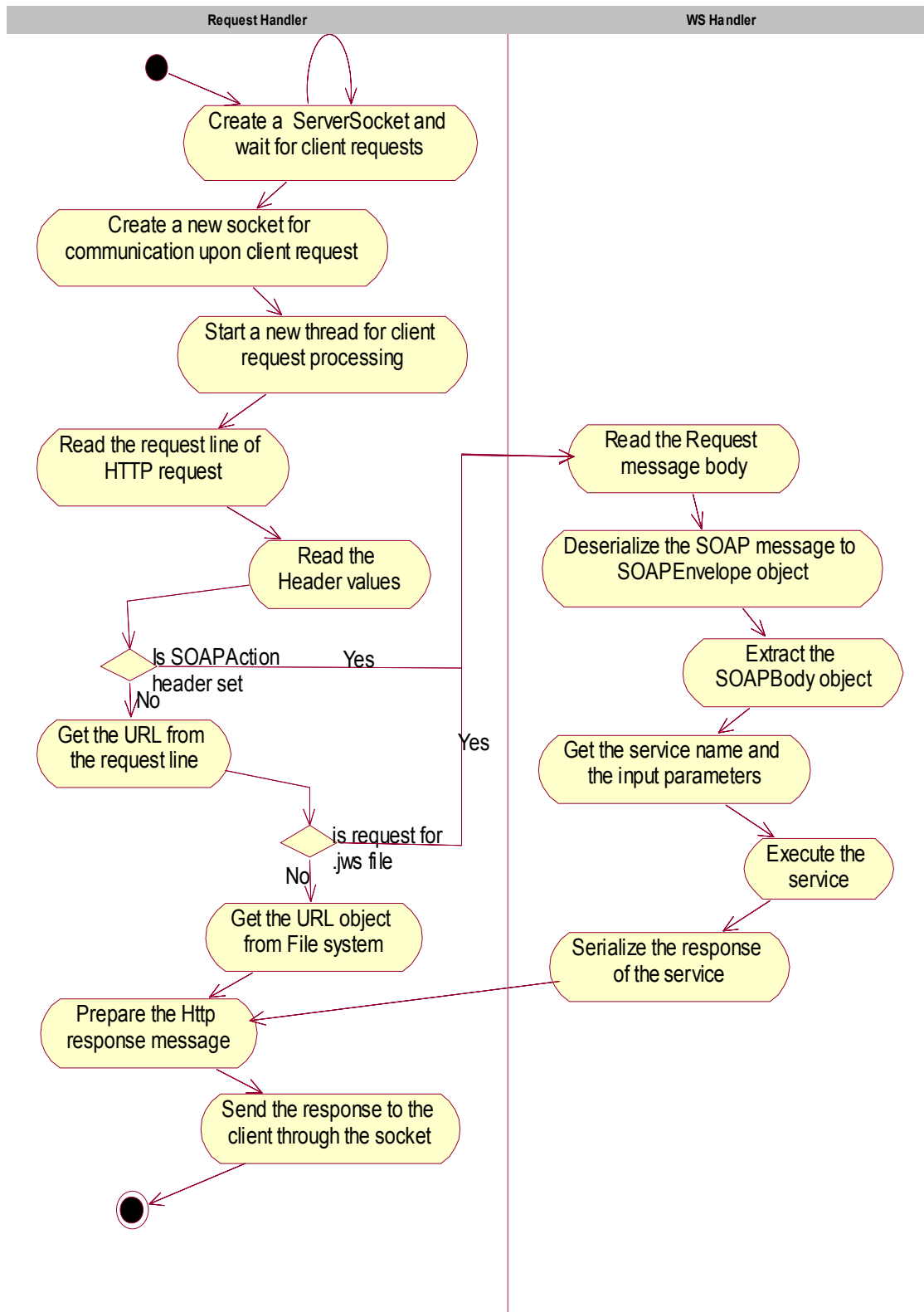


Figure 5.5: Activity diagram showing the core Mobile Host functionality

5. “SSNServer” – Mobile Web Service Provider

Considering the low-resource constraints of the Smart Phones, the Mobile Host’s targeted devices, no deployment environment was provided. Hence, all services have to be deployed at the installation of the Mobile Host. An alternative implementation would be, that the Mobile Host looks for services at different locations (that are previously specified at the mobile host before deployment) other than the main JAR location, where the services could then be deployed at runtime. More details are discussed later in this chapter.

5.4 Implementation details

This subsection gives implementation details of the mobile Web Service provider. Here we first discuss the Mobile Host features, their implementation, and then the developed services. As mentioned earlier, the Mobile Host was developed using PersonalJava.

5.4.1 General features of the Mobile Host

The mobile Web Service provider developed for the Smart Phones provides the following features:

- A standard Web server handling HTTP requests,
- A basic Web Service provider handling Web Service requests from any Web Service requestor using SOAP, passing the requests with HTTP tunneling,
- Capable of handling concurrent requests,
- Support for deployment of new services even after the deployment of the Mobile Host on the Smart Phone,
- Small GUI for basic server operations like start, stop, exit and services deployed,
- Capability to provide memory usage details of the Smart Phone,
- Support for the performance analysis.

5.4.2 Package hierarchy

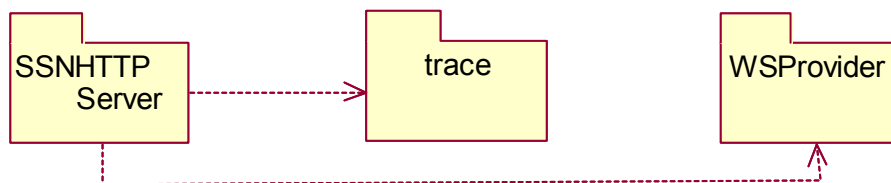


Figure 5.6: The packages of the Mobile Host

The functionality of the mobile Web Service provider is implemented in three packages, as shown in Figure 5.6. The classes implementing the basic Mobile Host functionality are in the “SSNHTTPServer” package. The “WSPProvider” contains the services provided by the Mobile Host. The “trace” package contains the utility classes used in the performance analysis of the mobile Web Service provider. The dependency relations between the packages are also shown in the figure.

The following subsections describe these packages in detail with respect to their functionality.

5.4.3 Mobile Web Service provider

As discussed earlier the main functionality of the mobile Web Service provider is implemented in the SSNHTTPServer package. The class diagram of the package is shown in Figure 5.7. An overview of the classes and their functionality is given below.

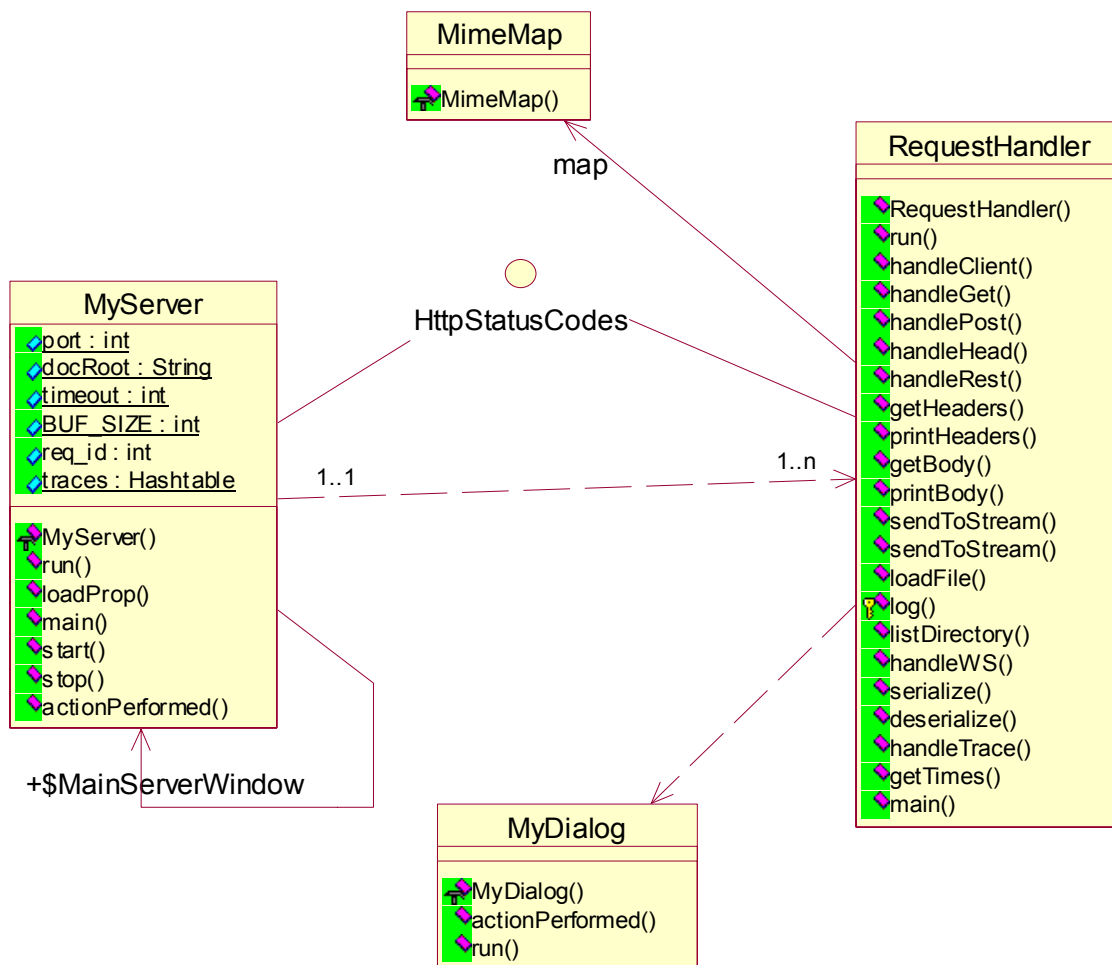


Figure 5.7: Class diagram of SSNHTTPServer package

MyServer:

This class sets the initial configuration required for starting the Mobile Host the document root directory of the server, port on which the server is waiting for requests, socket timeout, etc. Once these standard parameters are set, the server starts listening on the ServerSocket for incoming client requests and initiates calls to the RequestHandler object upon requests from the clients. The class also holds the trace details of each client request in a hash table to be used for the performance analysis.

5. “SSNServer” – Mobile Web Service Provider

Finally, the class provides the server with a Graphical User Interface (GUI) for restarting and shutting down the Mobile Host, and for the trace support. The GUI also provides the task manager support, which helps in the observation of the memory load on the Mobile Host

RequestHandler:

The RequestHandler is the main class, which implements most of the functionality of the Mobile Host. For simplicity, the request handler and the service handler functionality of the core architecture of the Mobile Host shown in Figure 5.4 are implemented in this class. The request handler creates a new thread for each client request of the Mobile Host and these threads handle the clients.

The thread extracts the data from the input stream of the socket, and the HTTP method of the request is identified. The thread then reads the header fields of the request, and checks for the SOAPAction header field. If the header value is not set, the thread processes the HTTP request just as the normal Web server, like returning a file, directory listing from the file system etc., and returns the response as standard HTTP response message by writing to the output stream of the socket.

If the SOAPAction header is set, the thread extracts the HTTP message body and deserializes the SOAP request to Java objects using the kSOAP and kXML API. After deserialization, the service name and the service parameters are read, and the respective service is called using the Java reflection¹ API. Once the service returns the result, the result is serialized and populated into the HTTP response message body, and the response is sent to the client by writing it to the output stream.

The class also supports the trace requests. If the request is for trace parameters then just the trace value is extracted from the static hash table object of the Mobile Host using the request id, and the trace data is sent to the client as the response.

MyDialog:

This is a utility class, which provides the GUI for the task manager support of the Mobile Host. As discussed earlier, the GUI displays the details of the memory usage details of the Smart Phone with respect to the JVM like the free memory and total memory of the server using the Java Runtime² support. For this the class uses the Runtime.freeMemory() and the Runtime.totalMemory() methods of PersonalJava.

MimeMap:

This class is a hash table holding the file extensions accepted by the server and their respective MIME³ types.

HttpStatusCodes:

This is an interface, which defines the standard response status codes of the HTTP protocol.

¹ The reflection API represents, or reflects, the classes, interfaces, and objects in the current Java Virtual Machine. More details are available at <http://java.sun.com/docs/books/tutorial/reflect/>

² The API of PersonalJava is available as Javadoc at <http://java.sun.com/products/archive/jdk/1.1/index.html>

³ MIME, Multipurpose Internet Mail Extensions, specification available at <http://www.mhonarc.org/~ehood/MIME/2045/rfc2045.html>

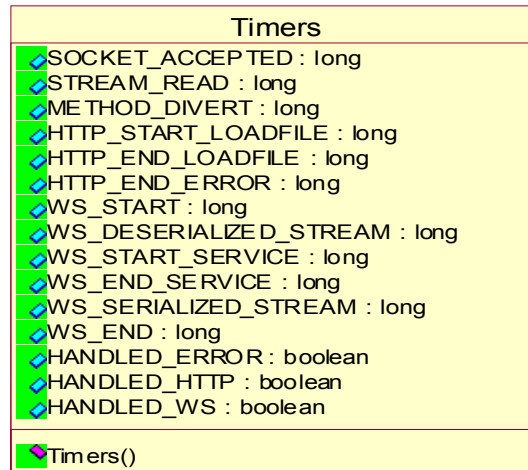


Figure 5.8: Timers class holding the trace parameters of the Mobile Host

As Discussed earlier, the trace package contains the utility classes used in the performance analysis of the Mobile Host.

The package has only one class, **Timers**, shown in the Figure 5.8, which holds the exact timestamps of the server at different instances of the Mobile Host execution with respect to each request of the server. These are the details stored in the hash table for future use in the performance analysis at the server as described earlier. Detailed description of all these variables is given in the performance study in chapter 6.2.2, where their description would be appropriate.

5.4.4 Services developed

The package WSPProvider holds the services provided by the mobile Web Service provider. As explained in the core architecture of the Mobile Host, the package can span across multiple JAR files, which are mentioned before the deployment of the server on the mobile terminal, where all, the server looks for the requested services. This is the package, which gives the Mobile Host the capability of “dynamic deployment of services”. The class diagram of the package is shown in Figure 5.9, with all the services developed for the mobile host as of now¹.

All the services in the figure are visible to the Web Service requester (client) as a single service provider with multiple public services, but as far as the implementation of the Mobile Host is concerned, each service is a new Java class, whose first method is provided as the Web Service. The name of the service matches with the name of a class in the WSPProvider package.

¹ Many other services were also developed, apart from the services shown in the figure, like the UserHandledGPS, which asks for the user’s permission before providing the GPS details, and etc.

5. “SSNServer” – Mobile Web Service Provider

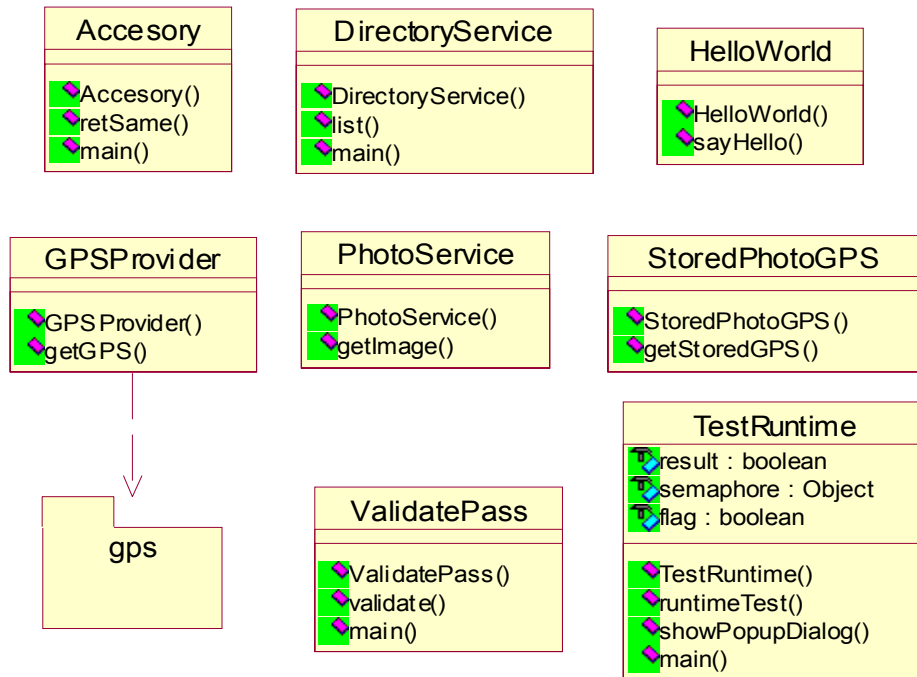


Figure 5.9: Class diagram of SSNHTTPServer package

An overview of each of the services is given below. Most of the services developed here are very small and generally two to three of these services contribute for any application development. Most of these services were required during the performance analysis for concluding different results.

PhotoService (mobile picture service):

The service provides the pictures taken by the Smart Phone. The service requires the name of the picture, under which it is stored in the file system of the Smart Phone, as parameter. The images are then looked up in the file system. The identified images are then Base64 encoded¹ into ASCII streams and are returned as a byte stream.

DirectoryService:

The service provides the directory listing of the server file system as a string. The string object can later be parsed to get the directory listing of the Smart Phone. For example, the service returns all the images in a directory, if the directory name is given as parameter. The response string can be processed later to get the different image names.

GPSPProvider (GPS provisioning service):

The service provides the GPS (Global Positioning System) location-based data. GPS² is a satellite-based system, operated and maintained by the U.S. Department of Defence (DoD). GPS provides accurate location and timing information to people worldwide. The system

¹ Base64 encoding is performed on the images, to transfer them using HTTP protocol, which supports only ASCII char streams. The Base64 encoding takes three bytes of data and represents them as four printable characters in the ASCII standard. More details can be found at http://email.about.com/cs/standards/a/base64_encoding.htm

² More details about GPS at <http://www.montana.edu/places/gps/1Basic/index.html>

transmits radio signals that can be used by GPS receivers to calculate position, velocity and time anywhere on earth, any time of day or night, in any kind of weather. The system has 24 satellites orbiting the earth, which give exact location of any point on earth. The system provides different positioning details like longitude, latitude, altitude, and the time etc.

So, for providing the location details by the Mobile Host, the service uses a Socket GPS receiver¹, connected to the Smart Phone via Bluetooth. An Ericsson proprietary BlueGpsLocator server component application written in c++ is installed on the Smart Phone. This BlueGpsLocator application continuously reads the GPS data from the serial port of the device, and returns the GPS data on port no 8180. The service on the Mobile Host uses a utility class GpsReader, which reads the data returned by the BlueGpsLocator application on port 8180, and returns the data as a GpsData object. The service extracts the location details like longitude, latitude, altitude, the speed, and the status of the device from the GpsData object, and returns these details as a String object.

The class diagram of the utility classes used in this service is shown in Figure 5.10.

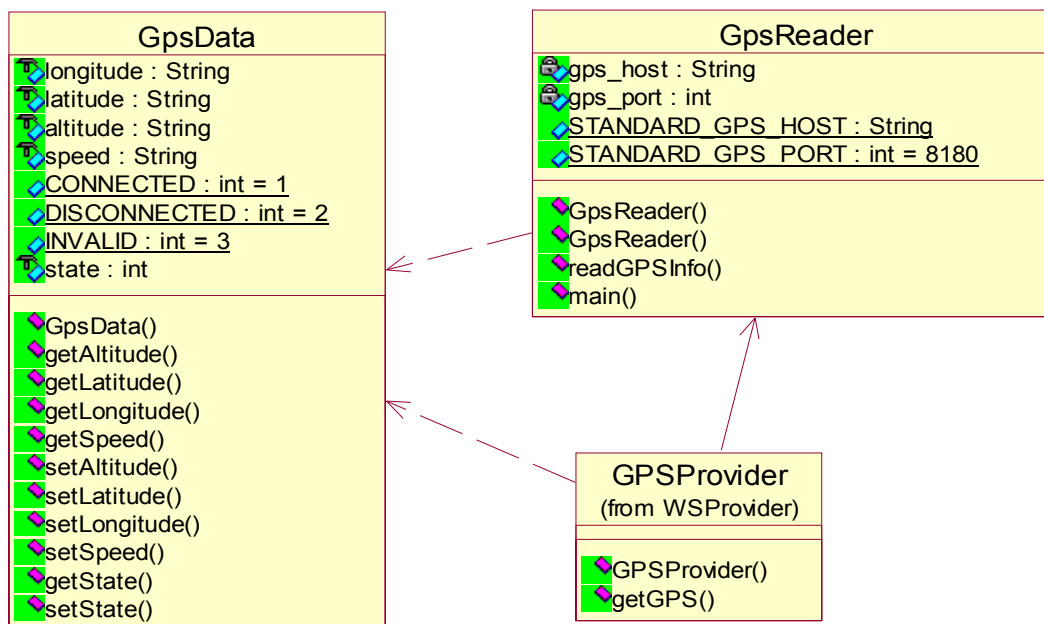


Figure 5.10: The utility classes of the GPSProvider service

StoredPhotoGPS:

The service returns the GPS data associated with the picture. The GPS data, i.e. the location where the picture was taken with the Smart Phone, is stored along with the picture. The service accesses these data and returns them to the WS requestor.

TestRuntime:

The service is used for switching between the tasks on the Smart Phone. The service uses the TaskSwitch² application installed on the Smart Phone.

¹ Information of the GPS device is available at <http://www.socketcom.com/product/GP0804-405.asp>

² The TaskSwitch is a Symbian proprietary application. More details about the product and its usage are available at http://www.symbian.com/developer/downloads/java_util.html

5. “SSNServer” – Mobile Web Service Provider

Accessory (Echo service):

The service echoes the string it receives as the input parameter. The service was required during the performance analysis. More details are given in the performance analysis in chapter 6.2.3.

ValidatePass:

The service is used for the authorization of the Web Service requestor. The service verifies the username and password sent by the client.

HelloWorld:

This was a basic service used for the initial testing of the Mobile Host. The service just greets upon providing the name.

5.4.5 Mobile Host GUI

The GUI for the Mobile Host developed and deployed on the Smart Phone is shown in Figure 5.11. It has support for stopping and starting the server. The GUI displays the status of the server and the deployed services on the mobile Web Service provider. The Mobile Host also has support to remove the traces parameters, so as to conserve memory resources, as and when necessary.



Figure 5.11: The Mobile Host GUI

Summary:

The chapter discussed the Java implementations available for the Smart Phones. Then it discussed briefly the HTTP protocol and message formats. It later discussed the architectural features of the Mobile Host and also addressed the implementation details and explained the developed components.

5. “SSNServer” – *Mobile Web Service Provider*

6 Performance analysis

This chapter describes the performance analysis conducted in the mobile Web Service environment with mobile terminals as Web Service clients and providers. It explains the conducted experiments and the analysed results in detail.

6.1 Mobile terminal as Web Service Client

In order to evaluate whether Web Services could be used from a mobile device, a simple request-response scenario is started, in which an application running on the mobile device requests some data from an application server on the Internet. The performance of a solution based on WS (SOAP) is compared with similar solutions based on other protocols such as Java RMI or an optimised protocol implemented directly over TCP. The following subsections give only a brief¹ description of the study as much research was already completed and this part of the thesis comprised only the study of the results.[18]

6.1.1 Test setup

The environmental setup for the mobile Web Service client experiment is shown in Figure 6.1. The mobile TCP/IP connection between the test client and the test server is deployed on top of a GPRS link into the mobile operator network. From there the traffic is routed through the Internet to/from the server used to measure the performance.

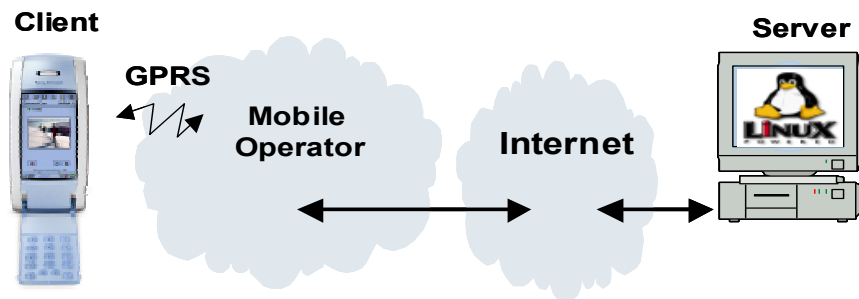


Figure 6.1: The experimental setup for mobile Web Service client

All tests were performed over a live GPRS network² (Vodafone D2 in Germany). For reference purposes, the same measurements as performed via the mobile connection have also been performed via a fixed client-server connection (Ethernet LAN). The following protocols were compared:

- SOAP over HTTP
- SOAP over HTTP with compression
- SOAP over TCP
- Java RMI
- Lightweight protocol running over TCP

¹ The proprietary information of Ericsson, where the experiment was performed, deprives me from giving detailed description

² Generally this is the GPRS connection used throughout the tests explained in this chapter

6. Performance analysis

The protocol, SOAP over HTTP with compression, offers an optional compression using the gzip algorithm.[15][16] The SOAP over TCP is implemented only on the fixed link, as the VB/C++ implementation is only for the full Windows platform.

Most of these protocols support persistent or non-persistent TCP connections. When persistent connections are used, the same TCP connection is re-used for all requests and responses within the same session. In the non-persistent case, a new TCP connection is opened and closed for each request-response pair. Both cases were tested for the protocols that supported these options.

6.1.2 Test cases

In order to have a stable base for the analysis, all requests have the same size. Each request contains three parameters: one integer, one string and one floating-point number. These parameters describe what the client requests from the server. They determine respectively the number, type and size of the objects returned by the server.

Depending on the parameters of the request, the server's response will contain the number of objects (1, 10, 100 or 1000) of a given type (string, integer or float) and of a given size (1, 10, 100 or 1000 characters for the strings and 1, 4 or 10 digits for the integers). These combinations of sizes and types allow us to have a good coverage of the basic data types that could be included in a message. The results obtained from these test cases allows to interpolate or extrapolate the corresponding values for messages containing almost any number of integers or strings of any length.

However, some combinations of parameters were excluded for the tests over GPRS. For example, sending a response message containing 1000 strings of 1000 characters would have generated too much traffic, especially since each measurement has to be repeated several times in order to have statistically valid results. So for optimal results, the following test cases were measured for all protocols over GPRS:

- Empty response (0 objects)
- 1 string of 1 character
- 1 string of 10 characters
- 1 string of 100 characters
- 1 string of 1000 characters
- 10 strings of 10 characters
- 10 strings of 100 characters
- 100 strings of 10 characters
- 1000 strings of 1 character
- 1 integer of 4 digits
- 10 integers of 4 digits
- 100 integers of 4 digits

6.1.3 Experimental results

The following subsection gives a brief description of the experimental results. The test case "10 strings of 100 chars" is considered to be the optimal case and its results are described in the following subsections.

6.1.3.1 Evaluation of the exchanged data

The following three charts (Figure 6.2, Figure 6.3, Figure 6.4) show how the total size of the response evolves according to the size of the object transmitted or according to the number of objects.

The first chart shows the size of the response containing one string of 1, 10, 100 or 1000 characters. The second chart shows the size of the response containing 1, 10, 100 or 1000 strings of 1 character each, while the third chart shows the sizes of the response containing 1, 10, 100 or 1000 integers of 4 digits. By interpolating these values, it is possible to estimate what would happen for messages containing an arbitrary number of objects of arbitrary sizes.

So by observing the charts it is clear that the results for SOAP are a little larger. Overhead added by SOAP is very significant, if the message contains a large number of objects. SOAP (which is based on XML) requires more bytes to encode its data compared to protocols using binary data, such as Java RMI. This overhead is due to the structural mark-up surrounding each object. So, Figure 6.3 shows that the size of SOAP response is very large when compared to other protocols for test case with response of 1000 strings with 1 char each. But the case is very rare in general applications.

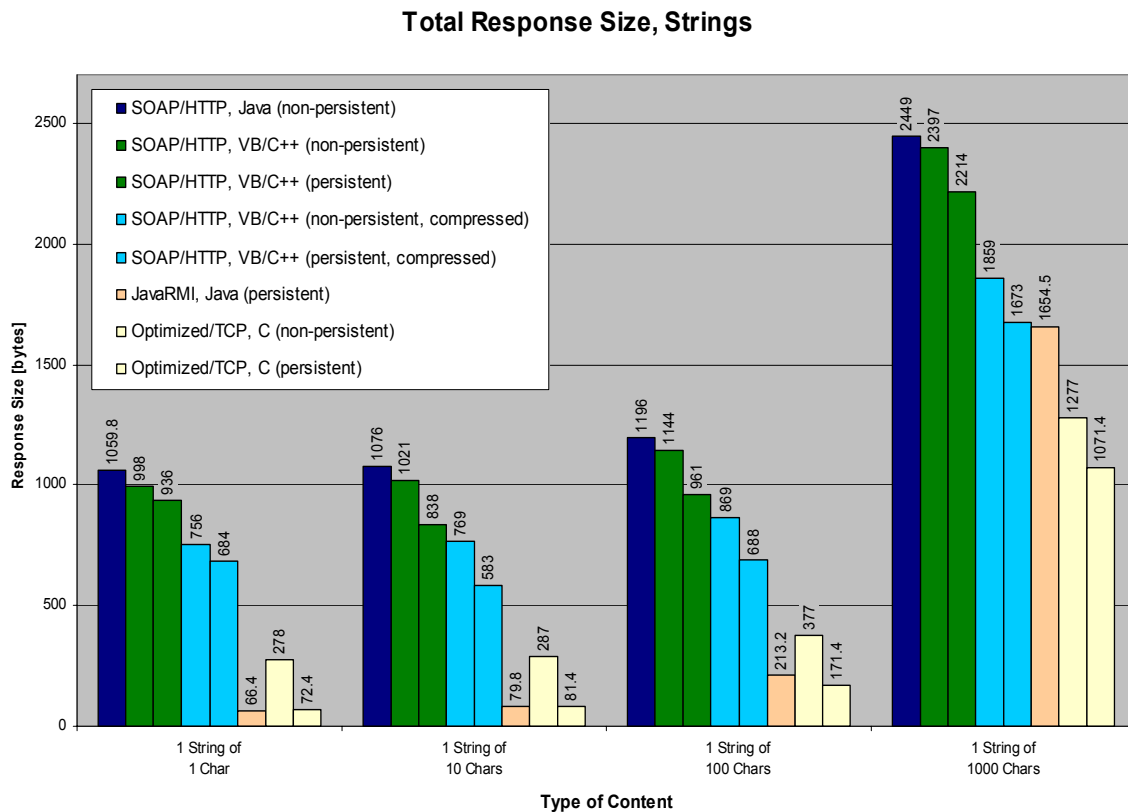


Figure 6.2: Response sizes for 1 string of different sizes

6. Performance analysis

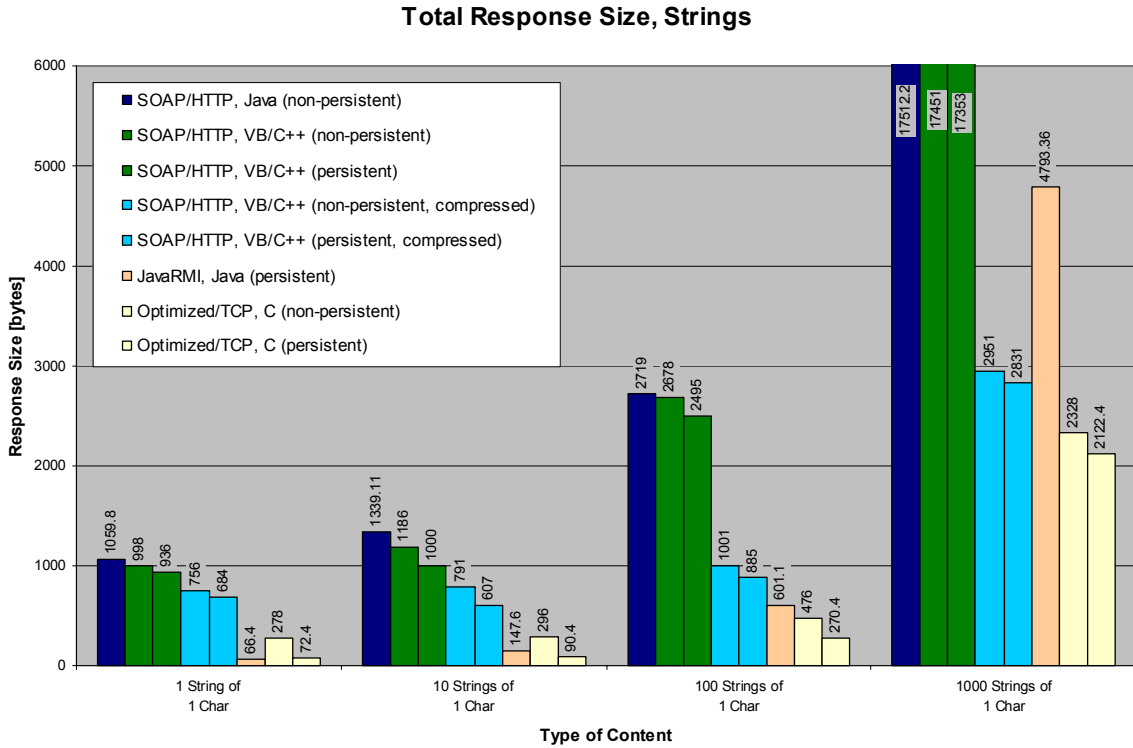


Figure 6.3: Response sizes for different no of stings of 1 char size

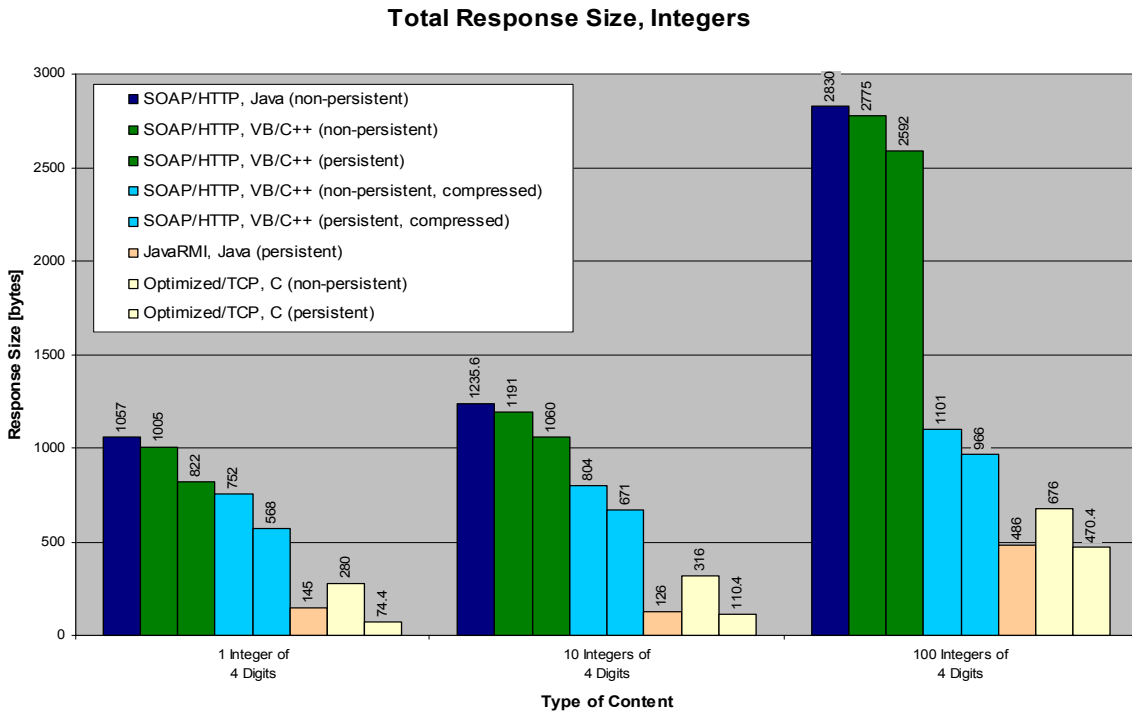


Figure 6.4: Response sizes for different no of Integers of 4 digits length

6.1.3.2 Response times

The total time for one complete round trip scenario comprises of the processing time for SOAP serialization and de-serialization and the transmission time. The following Figure 6.5 gives the time divisions for the optimal case “10 strings of 100 chars”.

The transmission time depends on the amount of data exchanged, but it also depends very much on the number of round-trips required for opening and closing TCP connections. The processing time for SOAP serialization and de-serialization was usually between 50ms and 200ms for the various types of messages that were tested. This means that the processing time for encoding and decoding SOAP messages is less than 10% of the total time perceived by the user.

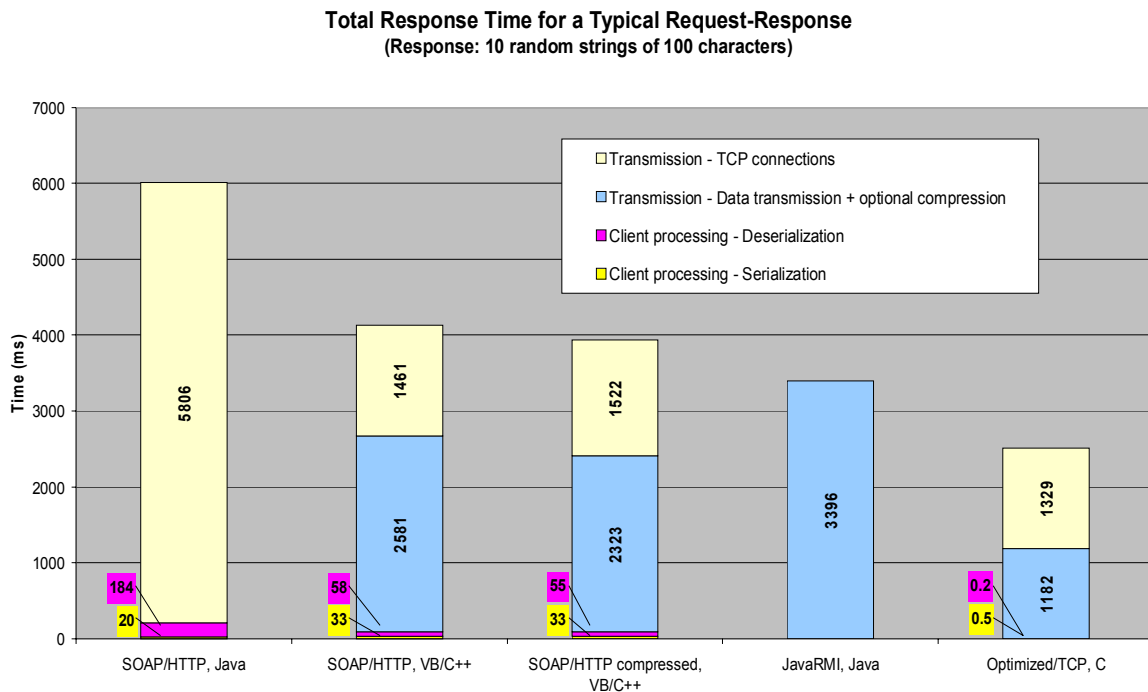


Figure 6.5: Total response times for the typical case

6.1.4 Observations

From the results observed in the previous subsection the following conclusions can be drawn regarding feasibility of Smart Phones as mobile Web Service clients.

- It is possible to invoke Web Services from a mobile device. SOAP is good enough because the overhead compared to other protocols is still small.
- The processing time is only a small fraction of the total time (< 10%).
- A significant part of this total time comes from the high latency of GPRS networks and the time taken for opening and closing TCP connections

6.2 Mobile terminal as Web Service Provider

In the chapter 5 we had discussed the implementation details of the Mobile Host. Once the Mobile Host was developed, it was extensively tested for performance issues like the memory load, server-processing load etc. The following subsections describe the experiments conducted and explain the results observed during this performance analysis of the Mobile Host.

6.2.1 Test setup

The test setup for the mobile Web Service provider's performance analysis is shown in Figure 6.6. The mobile Web Service provider is developed and deployed on the P800 Smart Phone. A standalone Axis¹ client (The client can be any standard Web Service client) program was developed, which accesses the Mobile Host as a Web Service requester. The client calls for different services deployed on the Mobile Host and the performance of the Mobile Host is observed, while the Mobile Host is processing the WS request. The client identifies and addresses the services deployed on the Mobile Host by different means, described in detail in chapter 3.2.2. The client used both the HSCSD and GPRS connections and the results were observed.

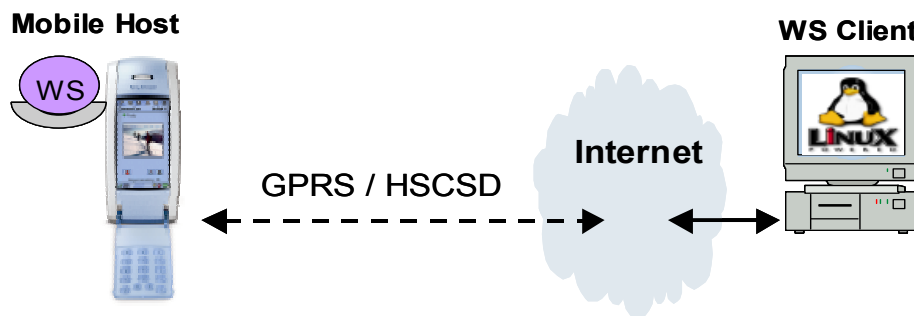


Figure 6.6: The test setup for mobile Web Service provider

6.2.2 Traces architecture

The main intension of these tests was to observe the time division of the total processing time of the Mobile Host for different activities of the server, like server processing time and transmission time etc., from one complete cycle of the request-response scenario. For this, timestamps² were taken at different instances of processing a request and these timestamps were observed for the delays caused by different activities later. Figure 6.7 shows different operations performed during one complete cycle of a request response scenario across the time axes.

¹ Apache Axis is an implementation of the SOAP, which can generate Web Service requests. More details of the implementation and API are available at <http://ws.apache.org/axis/>

² These timestamps are taken using the method `System.currentTimeMillis()` of `PersonalJava`. The method returns the difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC. So the precision of these timestamps is to milliseconds.

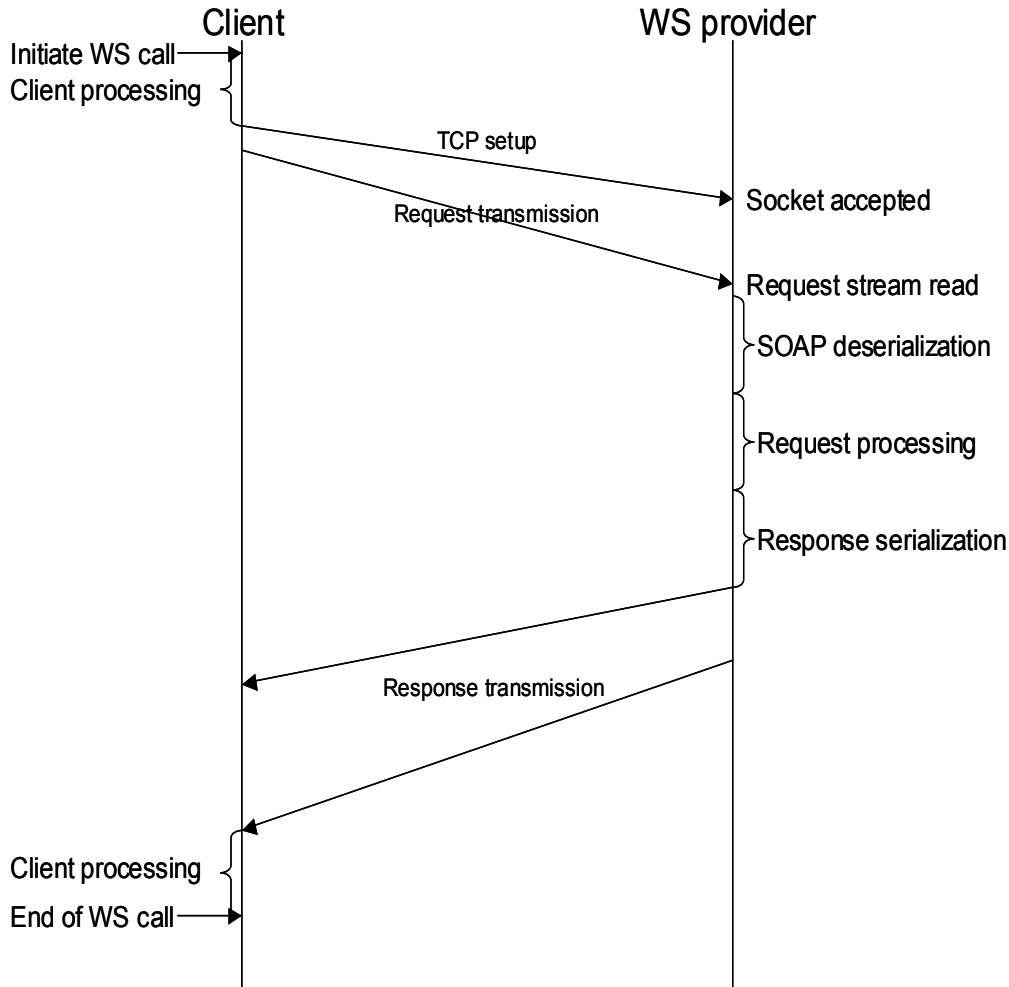


Figure 6.7: Operations performed during a request response scenario

The client initiates the call for a service and prepares the SOAP request, with details like the service name and input parameters etc., and then transmits the request to the mobile Web Service provider by establishing a TCP connection. The WS provider then extracts the SOAP request from the HTTP request message stream and deserializes the XML based SOAP request stream to KSOAP's SOAPEnvelope object. The WS provider then processes the request and populates the response. The response is then serialized to XML data streams and the response is transmitted to the client using the TCP setup already established. The client then processes the response stream from the Mobile Host and the actual response of the service is returned to the invoking client program.

The request and response messages are transferred to the WS provider in the form of TCP packets. So some delay could be caused for packet loss, TCP congestion control¹ etc. The delay is shown in the Figure 6.7 as the slanting lines for request and response transmission.

If the request is a normal HTTP request then the SOAP de-serialization and response serialization processes are bypassed at the mobile Web Service provider and the

¹ More details about packet loss, TCP Congestion control can be found at [3]

6. Performance analysis

communication is through standard HTTP request and response messages, as in standard web server, described in chapter 5.2.1.

During this complete request-response scenario, time stamps are taken at different stages of the process execution. These values are stored in a Java class (Traces) as shown in Figure 5.8 in the implementation details. The different instances, when timestamps are taken, are given below.

Timestamps are taken

- After socket acceptance at the Mobile Host
- After request stream extraction
- After method diversion, which decides weather the request type is a WS request or normal HTTP request.
- Before starting the Web Service request processing
- After the deserialization of the SOAP request
- Before starting the actual service (external¹ or internal)
- After finishing the service
- After serializing the SOAP response
- Before loading the requested file for HTTP requests
- After the HTTP response message is prepared
- After the response transmission at the server

The timestamps taken while processing a request are stored in a Hashtable at the Mobile Host, with a request ID, which is assigned for each request being processed. The Mobile Host can later be requested for these trace parameters and the results can be observed for any generalizations, at the client program. The client program, which generates the request, also has the total timing for the complete request-response cycle.

The following subsections describe the tests conducted using the test setup in Figure 6.6 and the observed results.

6.2.3 Experiments & results

This subsection describes the different tests conducted with mobile Web Service provider, developed and deployed on the Sony Ericsson P800 Smart Phone. The tests were conducted with different services discussed in chapter 5.4.4, 'Services developed'. The main services used for testing are the Mobile picture service and the GPS provisioning service.

The two² services are selected for testing because of their specific uses for the performance analysis of the Mobile Host. The first service, the Mobile picture service, returns the pictures taken by the camera that is integrated in the Smart Phone. For this service, the response size is very large (approximately 40kb) and this gives a large scope for the observation of effects of different parameters like transmission delays, the encoding performed on the response messages, the actual service delay etc. on the performance of Mobile Host. The second service returns just a small string containing the GPS data as the response, which gives

¹ The service that is accessing external devices for the service provisioning like the GPS provisioning service which uses a Socket GPS receiver connected to the Smart Phone via Bluetooth

² Apart from these two services, many other services were used for performance analysis, like the Accesory and DirectoryService etc. The services are mentioned while discussing the respective tests.

the scope for observing the behaviour of the Mobile Host under concurrent requests from multiple clients, there by observing the robustness of the Mobile Host under the multithreaded scenario.

All of the experiments were repeated several times¹ in order to have statistically valid results and the average values are analysed, mostly².

Test cases for Mobile Picture service

As explained earlier, the Mobile Picture service provides the pictures taken by the Mobile Host. So for initial testing 15 different images were considered as test cases with sizes ranging from 3kb to 93kb. The different images considered and their sizes are shown in Figure 6.8. The images are shown in Appendix A.

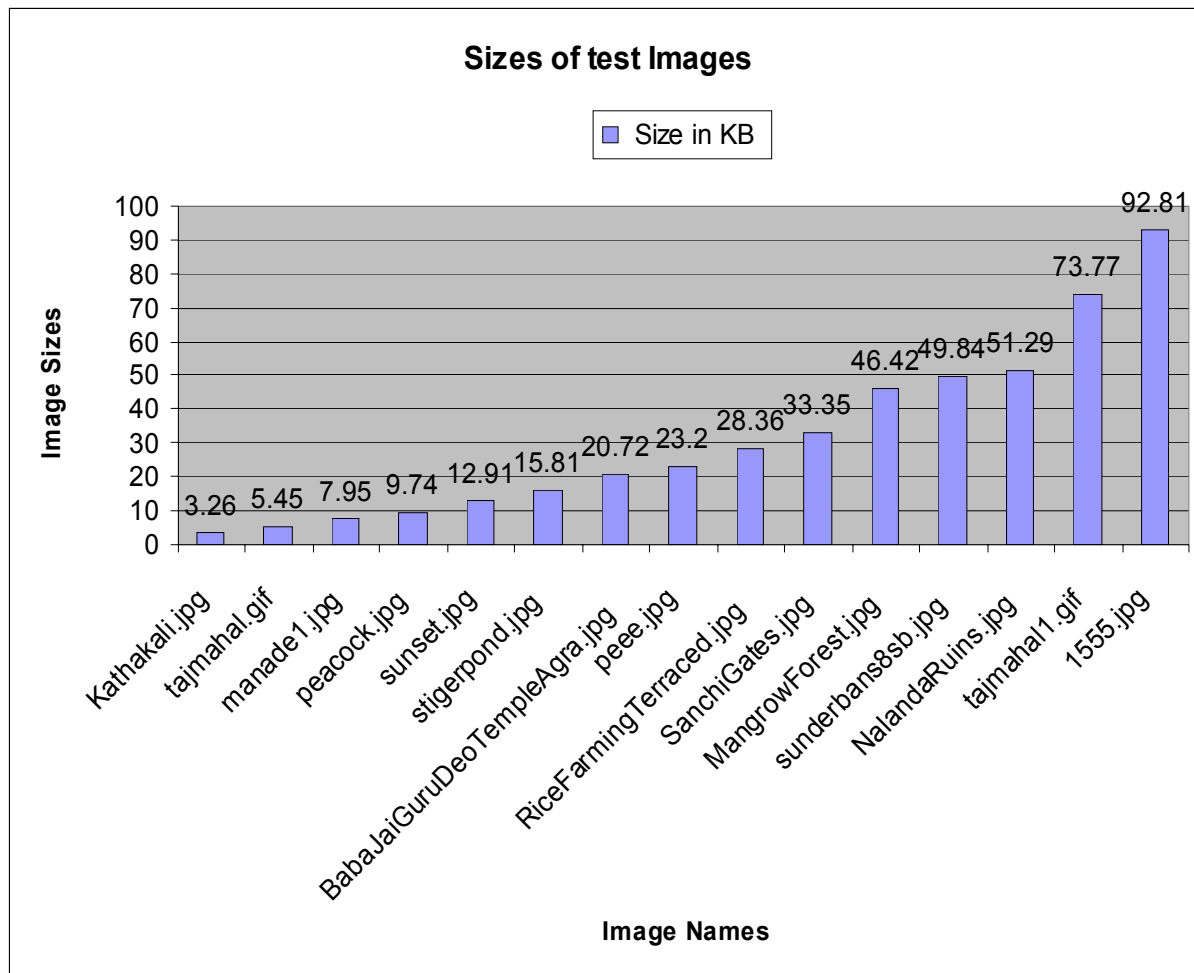


Figure 6.8: Test cases for the Mobile Picture service

These pictures are generally¹ used as the test cases for all those tests conducted with the Mobile Picture service, which are explained in the following subsections.

¹ Most of the experiments were repeated 5 times.

² Especially for the Mobile picture service, the number of test cases is 15 and each experiment is repeated 5 times, adding to 75 cases for an experiment. So, some of the experiments were analyzed with the best experimental results instead of averages.

6. Performance analysis

Comparison of WS and HTTP responses

The Mobile Host was first tested with the mobile picture service for calculating the SOAP processing delay of the server. For this the client generated requests for the test case images shown in Figure 6.8, both as normal HTTP client and also as the standard Web Service client. The difference between the two timings was calculated. The test was conducted using both the HSCSD and GPRS connections explained earlier.

The results showed a significant difference (approximately 20-25 % less time taken for the HTTP request) between the two timings. The results for the test conducted with the HSCSD connection are shown in Figure 6.9. The Figure 6.10 shows the SOAP processing % as the difference between the two values.

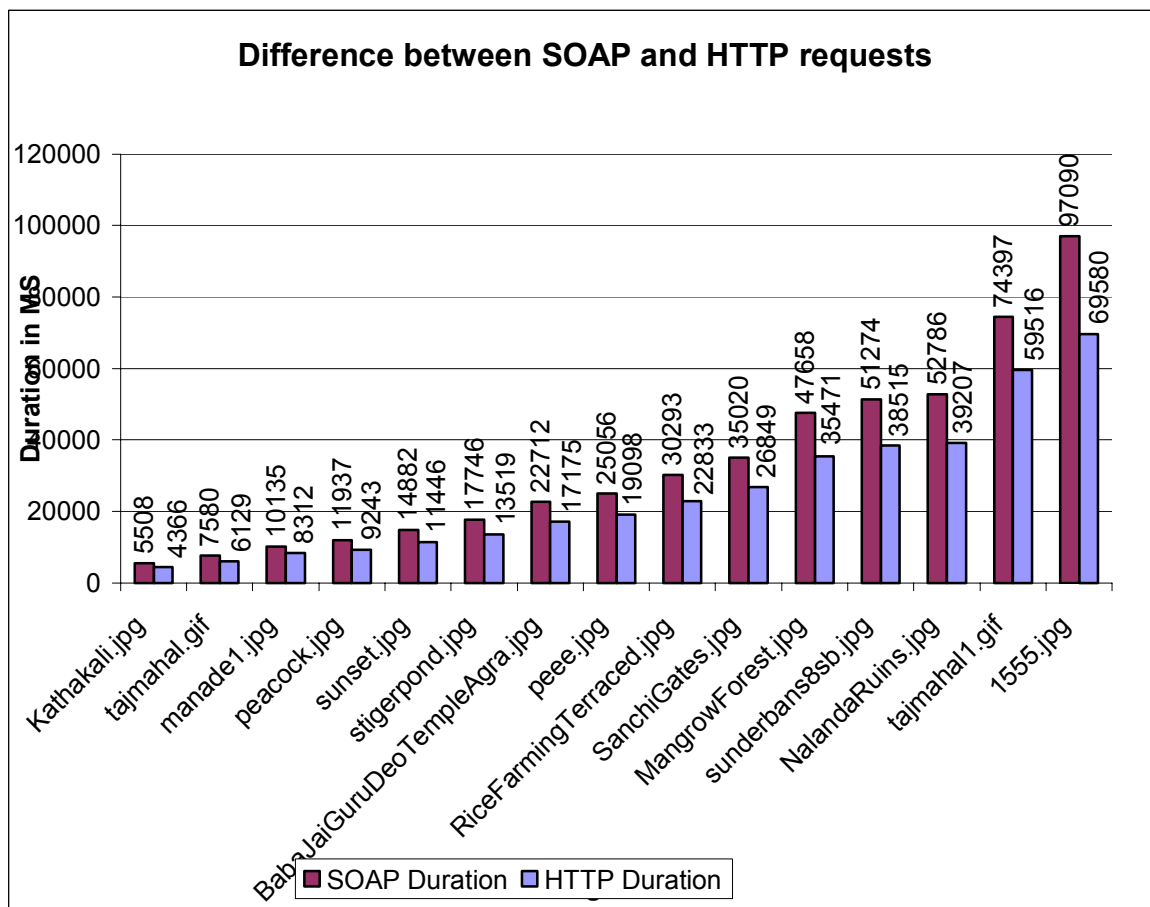


Figure 6.9: Difference between the SOAP and HTTP requests

¹ Some of the test cases like the images with 92.81 kb size are eliminated in some of the experiments like the regression test where the server needs to process multiple threads, since this would have generated too much traffic, especially since each measurement has to be repeated several times in order to have statistically valid results.

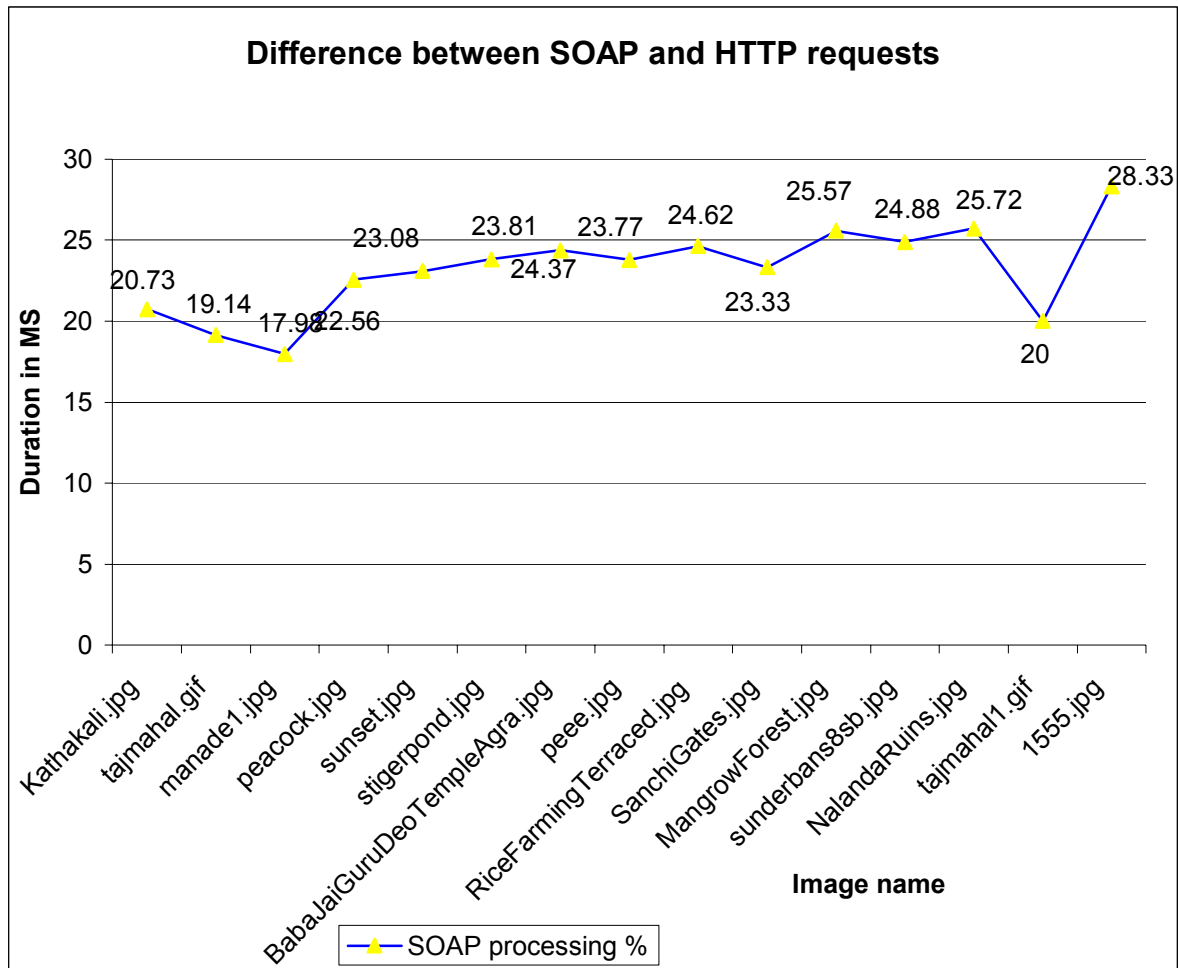


Figure 6.10: Difference between the SOAP and HTTP requests - 2

In the initial study of the thesis, it was assumed that the difference between the WS and HTTP timings was the actual SOAP processing delay of the Mobile Host. But later study revealed that the SOAP overhead and the Base64 encoding performed on the images before serialization and transmission of the response, has caused the size of the response to increase by more than 50%, which caused an increase in the transmission delay of the response, there by increasing the total time taken for the WS request processing at the Mobile Host.

The actual SOAP overhead caused to the size of the response is observed to be only 578 bytes. To observe this, the client requested the Mobile Host for the Accessory service (Echo service) explained in chapter 5.4.4, with single character as parameter. The service echoed the character by encoding the result in SOAP envelope. The size of the total response stream was observed at the server, just before transmission.

Delays caused by different activities on the Mobile Host

In order to identify the actual times taken for different activities on the Mobile Host like SOAP deserialization, serialization, transmission etc., the mobile picture service was requested by the client and the timestamps were taken as explained earlier. These timestamps were later processed to get the delays caused by these activities at the Mobile Host. The test is conducted

6. Performance analysis

using both the HSCSD connection and GPRS connection. The following Figure 6.11 shows the time delays of different activities for the test case images in Figure 6.8, using the GPRS connection.

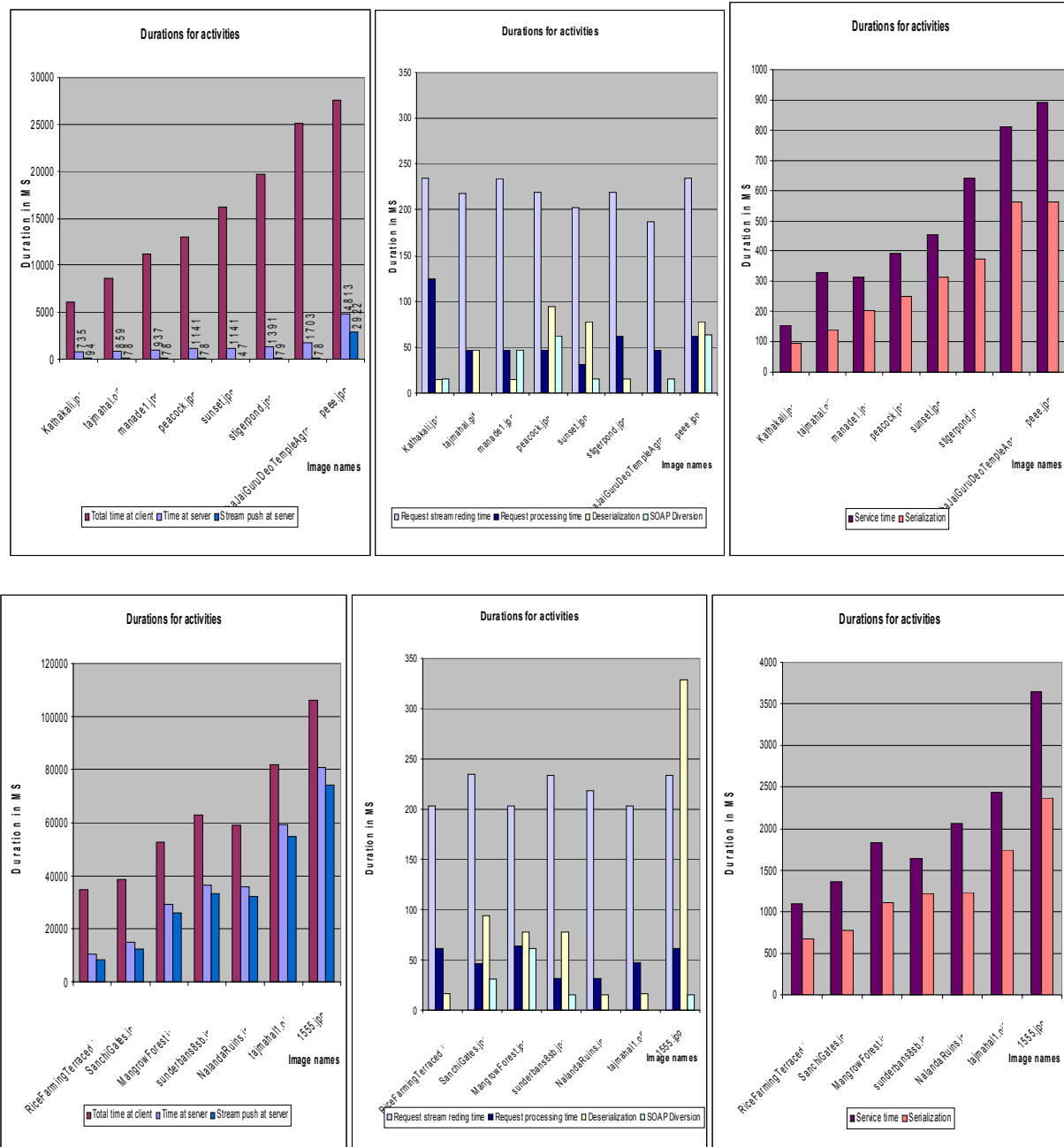


Figure 6.11: Time delays of different activities of the Mobile Host processing

The time delays shown in Figure 6.11 were later analysed with the test case image sizes, so as to get the dependence of these delays with the image sizes. The results are shown in graphs in Figure 6.12. The graphs clearly show that the total times at client and server, the stream push at the server, service time and serialization time are linearly increasing with the size and the rest of the delays were almost negligible and were mostly constants.

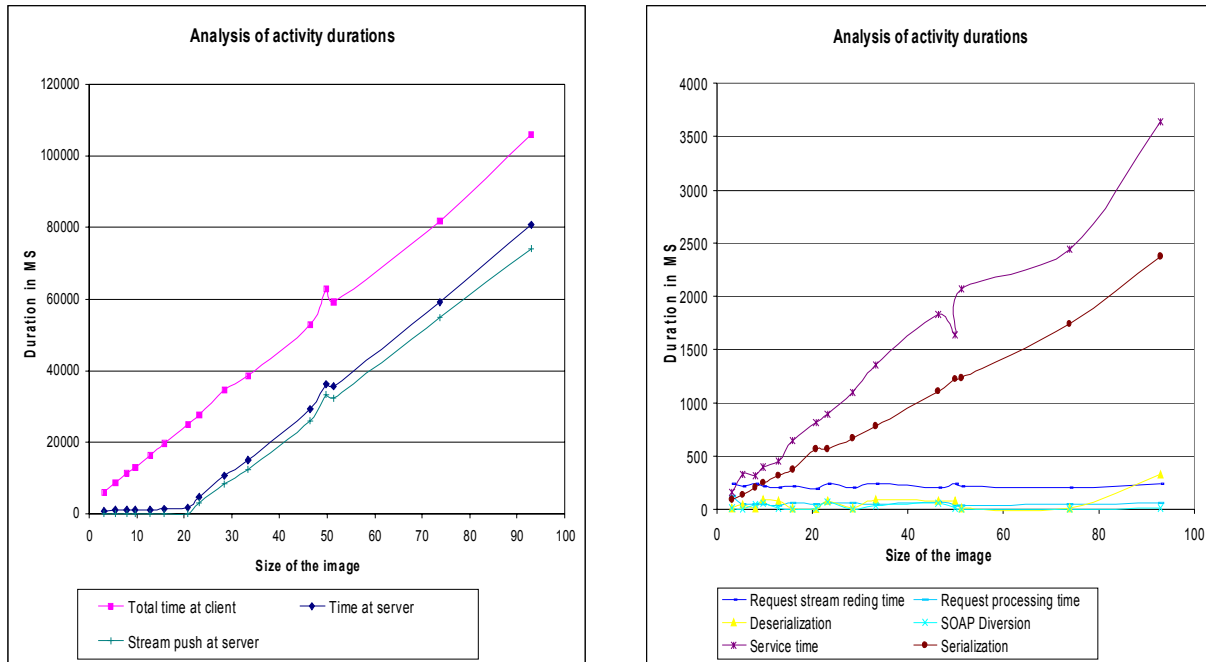


Figure 6.12: Analysis of activity durations with respect to the image sizes

The results of the test shown in the above Figure 6.11 suggest that time delays caused by the SOAP deserialization, SOAP processing and the Serialization are almost negligible when compared to the total processing time of the request. The results also suggest that most of the time at the server is actually the response stream transmission delay of the Mobile Host. So the total WS processing delay at the Mobile Host, as the combination of deserialization, SOAP processing and serialization, is calculated to be only 2.5% (approximately) of the total time of Mobile Host processing delay, rest all being the transmission delay. The Table 6.1 shows the WS processing delay % of different images with respect to the total processing time of the request.

6. Performance analysis

Image name	WS processing %
Kathakali.jpg	2.06
tajmahal.gif	2.18
manade1.jpg	2.37
peacock.jpg	3.11
sunset.jpg	2.5
stigerpond.jpg	1.98
BabaJaiGuru~1.jpg	2.31
peee.jpg	2.55
RiceFarming~1.jpg	1.98
SanchiGates.jpg	2.34
MangrowForest.jpg	2.37
Sunderbans8sb.jpg	2.08
NalandaRuins.jpg	2.11
tajmahal1.gif	2.13
1555.jpg	2.56

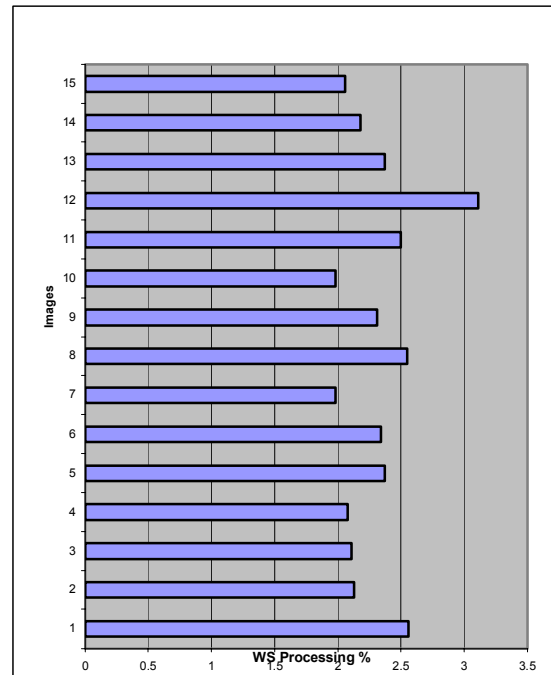


Table 6.1: WS processing % of the total processing time of the request

The tests described above all showed that the total SOAP processing time of the Mobile Host is very small (almost negligible with respect to the total time). All the above experiments were conducted with mobile photo service, where the transmission time was very high because of the large (approximately 60 kb) response stream for the service. So to observe the Mobile Host's performance with reasonable small services the following tests were conducted with GPS provisioning service. The response of the service is few KB and the time delays of different activities can be observed clearly.

With the GPS provisioning service explained earlier, the Mobile Host returned the GPS location based data of the mobile terminal as a String. The test was conducted using both the HSCSD connection and GRPS connection for addressing the mobile terminal from the client. The following Figure 6.13 shows the observed time delays for different activities at the Mobile Host for the GPS provisioning service. The results shown in Figure 6.13 are for the GPRS connection.

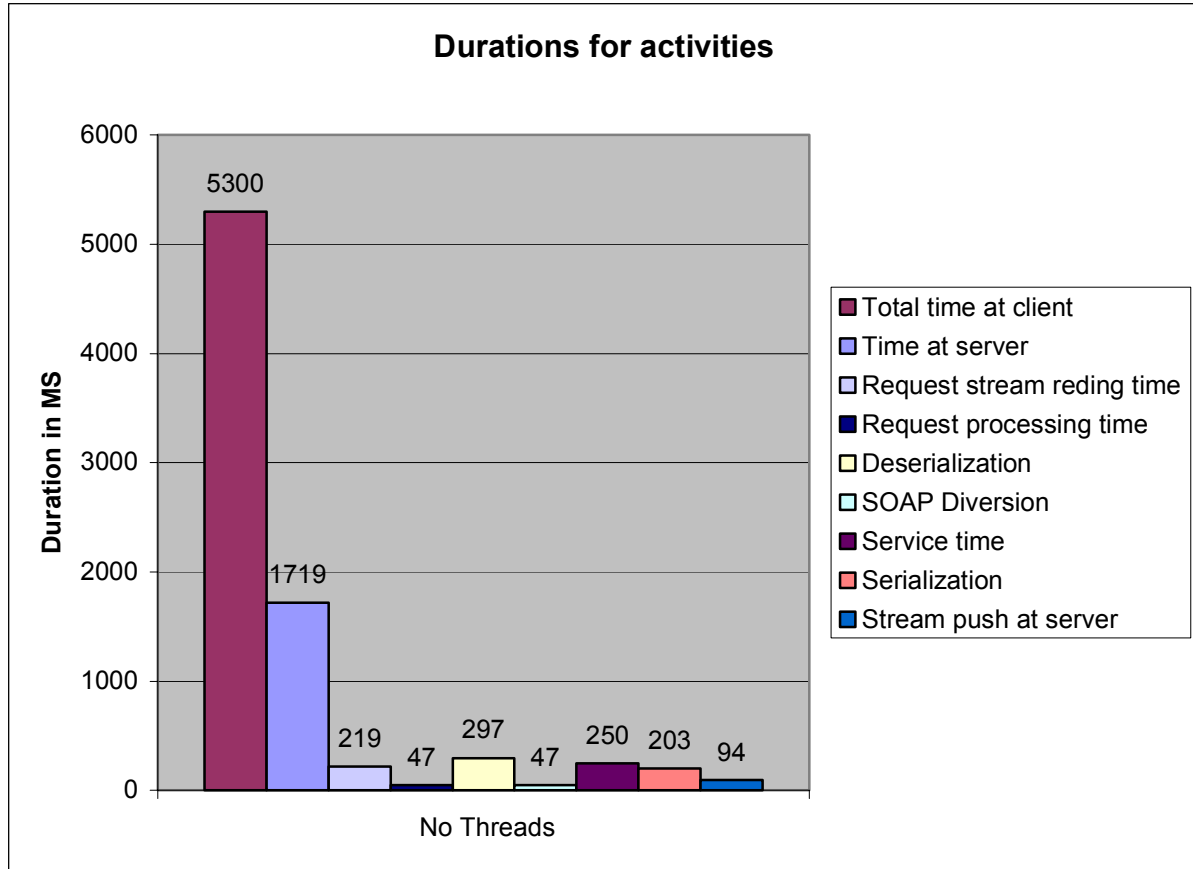


Figure 6.13: Time delays for the GPS provisioning services

The results showed in the Figure 6.13 suggest that the total SOAP processing time at the Mobile Host is still very small when compared to total request processing time. The mobile Web Service provider took 5.6%, 3.8%, and 10.3% of the total time for the deserialization delay, serialization delay and total SOAP processing of the Mobile Host respectively.

Concurrent access of the Mobile Host

Until now we had discussed different tests conducted with the single threaded case, where only one client is accessing the mobile Web Service provider at any particular instance. Only after a Web Service request to the Mobile Host is completely processed, the client generates another request.

Once the Mobile Host was successfully tested with the single threaded case, whose results showed that the actual processing delay caused by the Mobile Host is very small with respect to the total time, the service provider was subjected to regression testing with parallel clients.

For the regression testing, different WS clients concurrently accessed the Mobile Host for the deployed services. For that a client program was developed with Java threaded¹ support,

¹ A thread is a process of execution in a program. Java has extensive support for multithreading. The JVM allows an application to have multiple threads of execution running concurrently. More details at <http://www.java.sun.com>

6. Performance analysis

with each thread requesting for a service in parallel to other threads. The basic scenario of the test is still the same as shown in Figure 6.6.

The test was conducted with both the connection types, HSCSD and GPRS. The client accessed for the mobile picture service, with the test case images shown in Figure 6.8, by generating parallel threads. Each thread requests for all of the 15 images of test case images by generating WS requests one after the other. The test was successful with parallel threads up to 3 threads. For more number of threads the test failed. Further study of the results revealed the reasons for the failure of the test with more than 3 threads:

- Because of the threads, some requests failed as the Mobile Host could not accept sockets for those requests, which were generated simultaneously (For example 4 simultaneous requests generated and the server could accept only 2 sockets at that time. So the other two requests failed). Keeping a pool for the incoming requests and processing sequentially using LIFO¹ (Last In First Out) or FIFO² (First In First Out) algorithms could eliminate the problem. But this would increase the load on server just to maintain these request pools. In General, the client would generate a new request, if the request is not processed in some standard time, there by increasing the number of unnecessary requests in the pool, and there by affecting the performance of the server.
- Each socket is set with timeout of 5 seconds. With this parameter set for the socket, a read call on the Input Stream associated with this Socket will block for only this amount of time. If the timeout expires, a `java.net.SocketTimeoutException` is raised, though the Socket is still valid. So which ever thread could not start its output stream writing in 5 seconds terminated the connection with the client and the request failed for that test case. Removing the socket timeout could have eliminated the problem, but this could have made some of threads just wait for the read (IO) requests and would never terminate. This would have given us invalid results during the performance study.
- Some requests for the Traces also failed as they were also racing with the test case thread requests. Generating the Trace requests at the end of the test could have eliminated the problem. But as discussed earlier the traces are stored at the server until requested. So this would increase the memory load at the server and there by would affect Mobile Host performance. Also, once the Trace is requested the Trace is removed form the server memory. So the Trace requests are generated along with the normal WS requests, of course at the end of each thread, as this would free some resources for the server.

The experimental results for the test with mobile picture service, had not given us a clear picture of the actual Mobile Host's server performance issues (like deserialization time, serialization time etc) under the regression test, as we had already discussed, the transmission time itself was taking more than 97% of the total request time for the single threaded requests. And this percentage increased under threaded case as most of the time the Mobile Host was just transmitting the data, so the other parameters could not be observed clearly. So for this, the test was repeated with the GPS positioning service. For this service the test was successful up to 10 parallel threads.

¹ The algorithm used in generic Stack data structure

² The algorithm used in generic Queue data structure

The following Figure 6.14 gives the time delays for different activities at the Mobile Host for the GPS provisioning service under multithreaded scenario, for the fastest thread of 10 parallel threads generating the GPS data request. The results shown are for the test conducted using the GPRS connection.

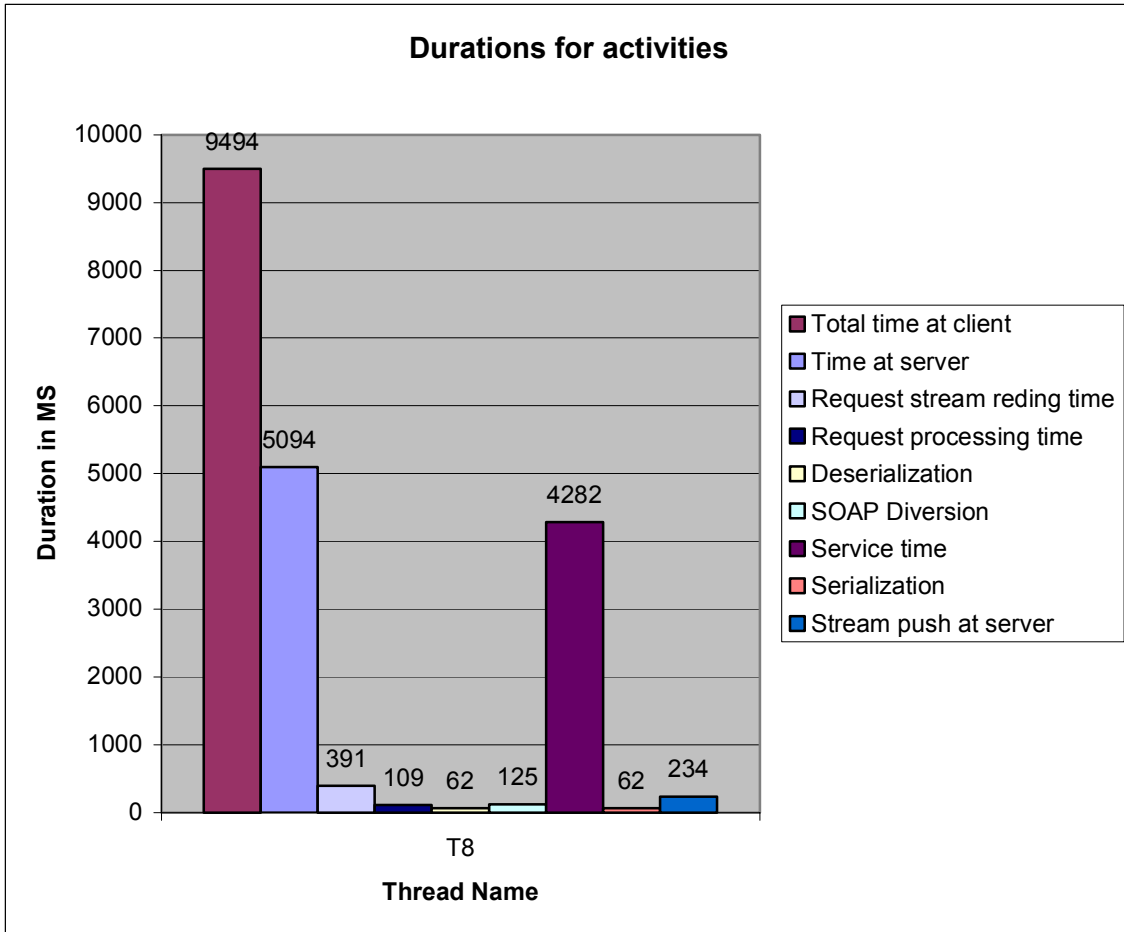


Figure 6.14: Time delays for the GPS provisioning services, shown for the fastest of 10 parallel threads

The results shown above, when compared with single threaded case results shown in Figure 6.13, suggest that there is not much difference for the time delays for different activities of the Mobile Host between the single threaded and the multithreaded cases. The only drastic difference being at the actual service delay i.e. the threaded case took 4282 milliseconds to access GPS data from the device, where as the single thread case took only 250 milliseconds. So the results suggested that the multi threading had affected the Mobile Host’s ability to access it’s resources like the file system and external devices to a great extent. The observation of the time divisions for all the 10 threads is shown in Figure 6.15.

6. Performance analysis

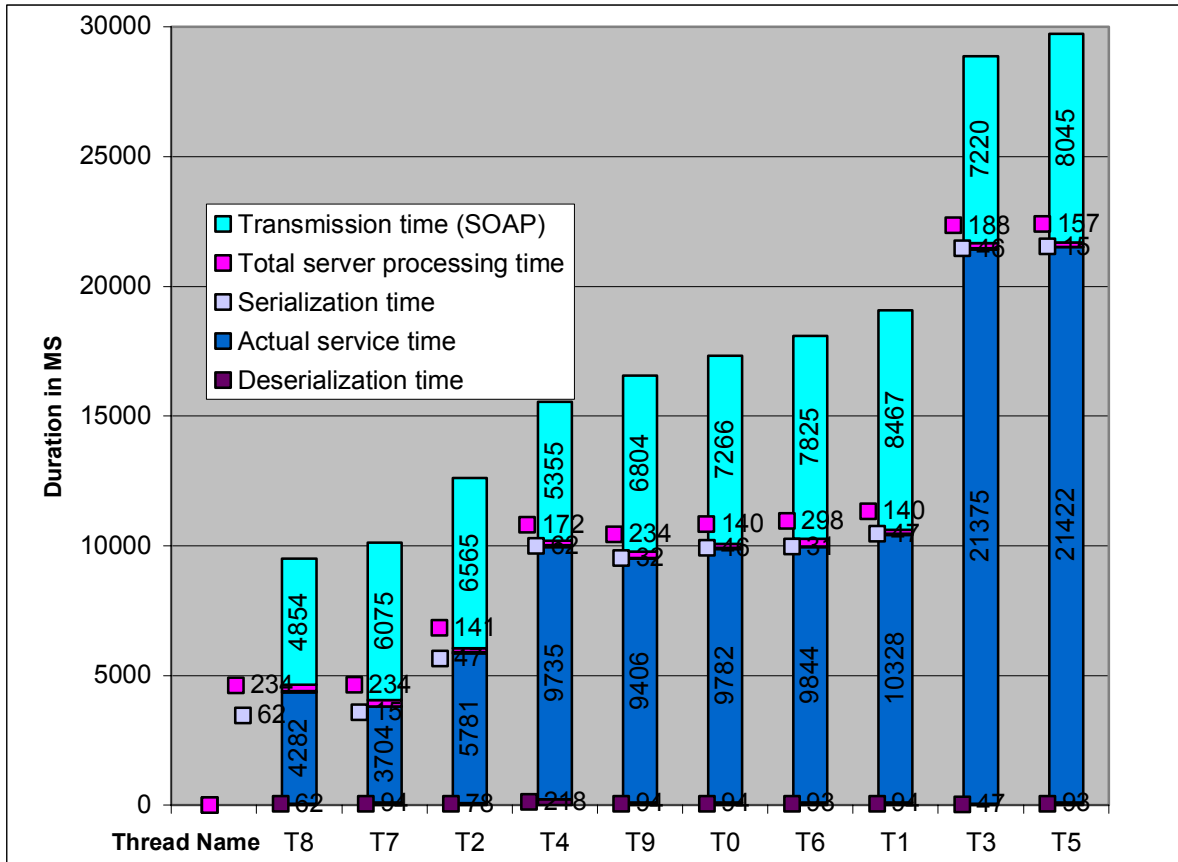


Figure 6.15: Time delays for the GPS provisioning services, 10 parallel threads

The observation of results in the Figure 6.15, gives more strength to the statement discussed above that the threads are affecting the Mobile Host's ability to access external devices. All the threads were just delayed because of the GPS provider device connected to the Mobile Host via Bluetooth.

The observation of the results also suggests that the mobile terminal maintains a queue for different threads for the access to external devices, as there were some sorts of jumps (observe first three threads, next 5 threads, last 2 threads) for actual service time, clearly shown in Figure 6.15.

For more than 10 threads, some of the requests failed at the Mobile Host. The reasons discussed for the mobile picture service are applicable also for the failure of this test.

Comparison of GPRS and HSCSD connections

As we had already discussed, most of the time of the total communication is for the transmission. So the server processing can be improved by increasing the bandwidth of the transmission. As discussed earlier with GPRS the transfer rate is more than with HSCSD. So to observe the affect of connection used on the total time for the request processing, the results of the mobile picture service test, for single threaded case, using both the GPRS and HSCSD connections are compared. The Figure 6.16 shows the comparison results.

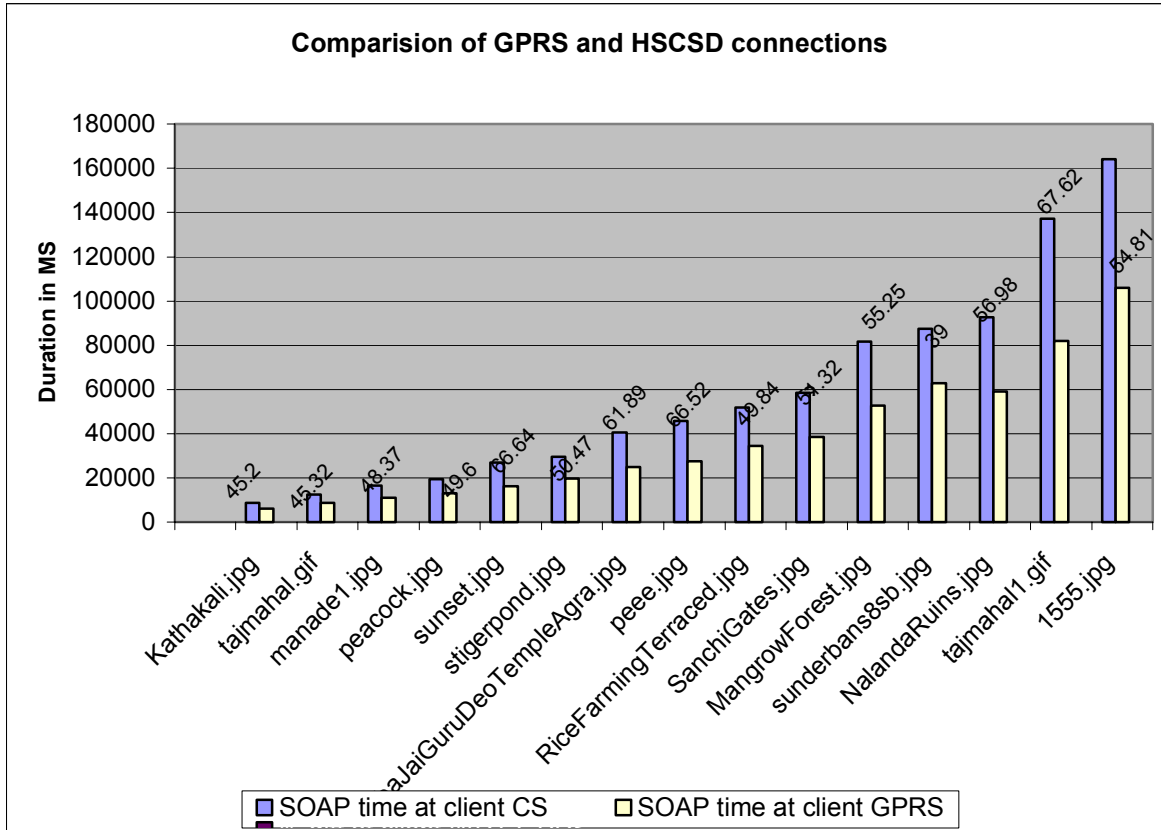


Figure 6.16: Comparison of total request processing time, for GPRS and HSCSD connections

The observation of above results suggests that there is a 55% (approximately) increase in total time for the request processing for the HSCSD connection with respect to the GPRS connection. The results were obvious as there was approximately 60% increase in transmission time, which contributes approximately 95% of the total time.

But the results showed above were sceptical, as the difference between the GPRS and HSCSD are not very significant as expected (Expected reduction in time in for GPRS connection to be 3 to 4 times less than that of HSCSD connection, as the HSCSD transmission rate is 28.8 kbps where as GPRS transmission rates can, theoretically, reach up to 144 kbps). So the tests were repeated many times predicting that the GPRS network to be busy as the same channels can be shared by many users.

The thorough study later revealed that the P800 Smart Phone, used for testing, is a “Type 4+1”¹ GPRS mobile phone, which means that it has four downlink channels and one uplink data transmission channel. So the transmission rate we were getting is only 1/5th (approximately, as there could other reasons effecting the transmission rate) of the total transmission rate possible.

¹ More details about GPRS phone types and transmission rates are available at http://www.cellular-news.com/gprs/what_is_gprs.shtml

6. Performance analysis

So, the best feature the Smart Phone had for ordinary case was a big liability for the study, obviously, as the scenario we are considering is complete reverse of general case, since the Mobile Host usually sends more data than it receives.

Memory load of Mobile Host

Apart from the time division for different activities of the total processing time of the request, during the thesis study, the memory load of Mobile Host was also checked. But, there was no generic tool support for Smart Phones, giving the details of memory usage of the Smart Phone at a particular instance.

Most of these tests discussed here, depended on the Java Runtime support for the amount of free memory and memory load of the Smart Phone. But this value is not a constant as the memory used by the JVM changes with activities like garbage collection¹ and as the sandbox of the JVM changes depending on the requirements. So at any instance we can get only the amount of memory that is free of the total memory used by JVM.

Before considering more details about the memory load of Mobile Host, we briefly discuss the behaviour of JVM memory management.

- When a JVM is invoked to run an application, it will ask the operating system for enough memory to run the JVM itself and some free memory for the application to create new objects and variables.
- When a new object is created, the JVM will allocate memory for that object out of the free memory area.
- When the free memory area is getting too small, the JVM will ask the operating system for more memory.
- When an object is no longer used by the application, it will be destroyed. Its memory will be freed up and merged back to the free memory area (Garbage collection).
- When the free memory area is used up, and there is no more additional memory available from the operating system, the JVM will stop the application and issue the "Out of memory error".

To get the memory load, the Mobile Host was provided with the task manager support, as discussed earlier in chapter 5.4.3, which gives the details of the memory usage (free memory and total memory) of the Smart Phone with respect to the JVM and these details were observed through out the span of different tests explained in this chapter.

The observation of these details suggested that, there was no problem for the Mobile Host with memory usage, as most of the time, the amount of free memory is at least 20% of the total memory allocated for the JVM and the "Out of Memory error" was never encountered during the execution of the tests.

¹ Garbage collection frees memory, by removing variables and finalizing the objects with out any reference in the JVM. More details about the Java Garbage collection at <http://www.javaworld.com/javaworld/jw-08-1996/jw-08-gc.html>

So, memory load was not a big problem for the performance of Mobile Host, as this is mainly handled by the JVM implementations provided for the Smart Phones and cannot be controlled directly.

6.2.4 Observations

From the results observed in the previous subsection the following conclusions can be drawn about the mobile Web Service providers.

- It is possible to have Web Services providers on a Smart Phone.
- The total processing time at the Mobile Host is only a small fraction of the total time (< 10%).
- The Mobile Host can process concurrent requests (approximately 10 parallel requests) for reasonable services.
- Concurrent access can affect the Mobile Host's ability to access internal and external resources.
- Increasing the transmission rates can increase the processing capability of the Mobile Host.
- The GPRS mobile phone type is affecting the maximum possible transmission rate for the Mobile Host, as the Smart Phones are allowing more transmission rates for downlink when compared to uplink.
- Generally, memory load was not a problem for the Mobile Host.

Summary:

The chapter described the performance analysis conducted in the mobile Web Services environment with mobile terminals as Web Service clients and providers. The chapter first discussed the research conducted with mobile terminal as Web Service client, where it explained the conducted experiments and analysed the results. The chapter then discussed the test setup for mobile terminal as Web Service provider and discussed the experiments and the results in detail. The chapter addressed different performance issues of the developed Mobile Host.

6. *Performance analysis*

7 Future research directions

The thesis study had successfully proved the feasibility of mobile terminals in Web Service domain as both Web Service providers and Web Service clients. This leaves us with a large scope for new research of Web Services in mobile domain and also studies of new domains for wireless networks. The following chapter gives a brief description of different areas, where the research can further be extended.

7.1 Proxy architecture

In the previous chapters, we had seen different means of communication with the mobile Web Service provider for the provided services. The architectures we discussed earlier gives scope for a proxy, which could be similar to normal HTTP proxy handling security, Authentication and congestion control etc. With the incorporation of such a proxy the operational setup of the Mobile Host would be as shown in Figure 7.1.

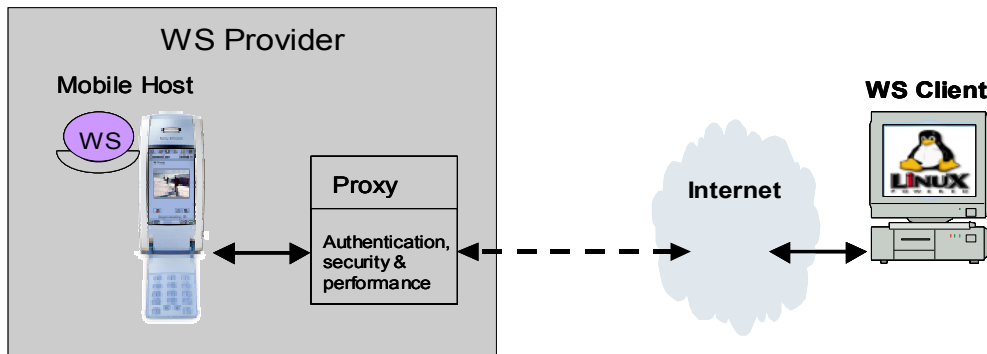


Figure 7.1: The operational setup of Mobile Host with proxy

The proxy can be programmed to handle security issues of the service provider like authentication, encryption etc., before passing the client request to the Mobile Host, thereby making the Mobile Host fool-proof even in the commercial environments. The proxy can also help in congestion control for example by allowing only a specific number of input requests at any instance, thereby increasing the robustness of the Mobile Host.

The proxy can be implemented either on the Smart Phone itself, or it can be implemented on some intermediate terminal as an individual component. In the first case the proxy behavioural issues can be deployed as services on the Mobile Host. In such a case the proxy services will be executed before handling the actual service request of the WS client.

The introduction of the proxy might require for new ways of identifying and addressing the Web Services deployed on the Smart Phone.

7.2 BEEP for SOAP message transmission

In chapter 2.1.2.3, while discussing the SOAP, we stated that SOAP messages could be exchanged not only with HTTP but also with other protocols like FTP and BEEP etc. The following Figure 7.2 clearly shows SOAP over different possible protocols.

7. Future research directions

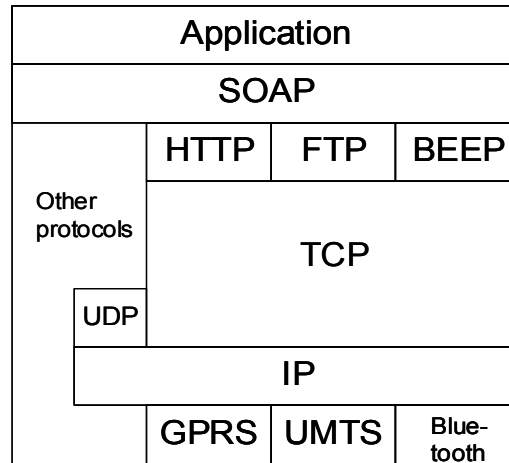


Figure 7.2: SOAP over different protocols

BEEP is an integrated collection of building blocks that gives you "best in class" data transmission solutions for everything from framing to security. Whether you're writing a simple "fetch" client/server application or a multi-threaded peer-to-peer relaying system, BEEP provides the necessary functionality without extra overhead. Different groups are already using BEEP for SOAP exchanges for desktop Web Services.

A good comparison of SOAP over BEEP and SOAP over HTTP is available at <http://www.xml.com/pub/a/2002/10/16/ends.html>. The thesis study mainly considered only SOAP over HTTP. The feasibility of SOAP over BEEP for the Smart Phones is to be checked. The specification for SOAP over BEEP is available at <http://www.ietf.org/rfc/rfc3288.txt>.

Others areas

The successful implementation of Web Service provider on Smart Phone also gives scope for new distributed architectures for Smart Phones, there-by making new payment methodologies, new modes of communication between the mobiles etc. in the wireless networks. Much research is yet to be done in this direction.

The thesis also suggested that much of the delay of a request processing is the transmission delay. The thesis identified some reasons for the transmission delay. Some research is still to be done, to eliminate these known problems or come up with different solutions for the Smart Phones used in Web Services domain.

The thesis introduced many applications with the Mobile Host. There could still be numerous possible applications of such a Mobile Host developed for Smart Phones, in different domains. Much research is yet to be done. For example, we had already discussed the Mobile Host's scope in mobile gaming domain, where the feasibility is yet to be checked.

Summary:

The chapter described different areas opening further research potential.

8 Conclusion

Within the thesis, the usage and feasibility of mobile terminals in the Web Services domain as both Web Service requestors and Web Service providers, was studied. The thesis also addressed different methods and architectures for identifying and addressing the Web Services deployed on the mobile Web Service provider, once it is deployed on the Smart Phone in a mobile network.

The thesis also realized a standard mobile Web Service provider (“Mobile Host”) for the Smart Phones. For the performance analysis of the Mobile Host, different services like mobile photo service and GPS data provisioning service were also developed. These services proved the feasibility of the Mobile Host even under concurrent access of multiple Web Service requestors.

The performance analysis of the Smart Phone, with both the mobile Web Service client and the mobile Web Service provider suggested that the actual Web Service Request processing time of the Smart Phone is only a small fraction of the total request processing time observed by the client.

The thesis also addressed the future research directions for the mobile Web Services.

The following items summarize the findings of the thesis.

- It is feasible to have mobile terminals in the Web Services domain as service clients.

The study addressed this issue and the performance analysis of such a Web Service client was compared with other standard protocols like Java RMI. The details are provided in chapter 6.1.

- It is also possible to deploy Web Services on the mobile terminal, i.e. it is feasible to have a Web Service provider on a Smart Phone.

To check the feasibility of this, the thesis realized a standard mobile Web Service provider (“Mobile Host”) for the Smart Phones. The implementation details of this Mobile Host are discussed in chapter 5.4.

- In both the cases, mobile Web Service clients and mobile Web Service providers, the total duration of processing at the Smart Phone is only a small fraction of the total request processing time.

The actual processing time at the Smart Phone is observed to be only a small fraction of the total time (< 10 %). More details are provided in the performance analysis of both mobile Web Service client and Mobile Host in chapter 6.

- The Mobile Host can also process concurrent client requests.

The Mobile Host realised during study, processed concurrent requests successfully. The details about the number of successful concurrent requests and the

9. Conclusion

reasons for failure beyond this number are explained in detail in “concurrent access of the Mobile Host” subsection of chapter 6.2.3.

- The Mobile Host on Smart Phones promises a new domain of applications, with applications possible in mobile gaming, transportation & logistics etc.

Chapter 4 discusses the possible use cases with the Mobile Host. The thesis also realised some of the applications discussed, like mobile photo album service and location data provisioning service. These services were used for the performance analysis of the Mobile Host.

- The Mobile Host paves way for new distributed architectures in the mobile environment.

The successful implementation of Web Service provider on Smart Phone also gives scope for new distributed architectures for Smart Phones, there-by making new payment methodologies, new modes of communication between the mobiles etc. in the wireless networks. Much research is yet to be done in this area. Chapter 7 discusses many other areas, apart from this, where the research can further be extended.

- The Web Services deployed on the Smart Phones can be addressed and accessed from Internet using different methodologies explained as in chapter 3.2.2.

Once a Web Service is developed & deployed with the Web Service provider developed on a mobile terminal, the mobile terminal, that is registered and connected within the mobile operator network, requires some means of identification and addressing, that allows the Web Service to be accessible also from outside the mobile network operator’s network domain. The thesis identified different methodologies and these are explained in detail in chapter 3.2.2.

LIST OF FIGURES

Figure 2.1: Basic operational relationships between Web Service components.....	9
Figure 2.2: Activity diagrams of service provider and service requestor.....	11
Figure 2.3: SOAP message structure.....	12
Figure 2.4: The structure of WSDL document	17
Figure 2.5: Relationships of UDDI data structures.....	21
Figure 2.6: The SoapObject structure of a parsed SOAP message[9].....	26
Figure 3.1: Basic architectural setup of mobile Web Service client	28
Figure 3.2: Basic architectural setup of Mobile Host.....	29
Figure 3.3: Architecture for an end-to-end TCP/IP connection between the mobile terminal and the prototyping network.....	30
Figure 3.4: The operational setup of Mobile Web Service provider in a live GPRS environment	32
Figure 3.5: The operational setup of Mobile Web Service provider.....	32
Figure 3.6: The architectural setup of Virtual mobile Web Service provider	34
Figure 3.7: The architecture for identifying IP at NAT translation	35
Figure 3.8: The architecture for identifying IP with session maintenance	36
Figure 4.1: The mobile photo album service scenario	39
Figure 4.2: The Mobile Host used in Journalism scenario	40
Figure 4.3: Scenario with Mobile Host in ad-hoc mobile gaming environment.....	41
Figure 4.4: Mobile Host used in traffic control systems	42
Figure 4.5: Guided parcel service scenario.....	43
Figure 5.1: Architecture of Java 2 Platform [7]	47
Figure 5.2: General format of HTTP request message.....	50
Figure 5.3: General format of HTTP response message	51
Figure 5.4: Core architecture of the Mobile Host	52
Figure 5.5: Activity diagram showing the core Mobile Host functionality.....	54
Figure 5.6: The packages of the Mobile Host.....	55
Figure 5.7: Class diagram of SSNHTTPServer package.....	56
Figure 5.8: Timers class holding the trace parameters of the Mobile Host.....	58
Figure 5.9: Class diagram of SSNHTTPServer package.....	59
Figure 5.10: The utility classes of the GPSProvider service	60
Figure 5.11: The Mobile Host GUI.....	61
Figure 6.1: The experimental setup for mobile Web Service client.....	64
Figure 6.2: Response sizes for 1 string of different sizes	66
Figure 6.3: Response sizes for different no of strings of 1 char size.....	67
Figure 6.4: Response sizes for different no of Integers of 4 digits length.....	67
Figure 6.5: Total response times for the typical case	68
Figure 6.6: The test setup for mobile Web Service provider	69
Figure 6.7: Operations performed during a request response scenario	70

List of figures

Figure 6.8: Test cases for the Mobile Picture service 72
Figure 6.9: Difference between the SOAP and HTTP requests..... 73
Figure 6.10: Difference between the SOAP and HTTP requests - 2..... 74
Figure 6.11: Time delays of different activities of the Mobile Host processing 75
Figure 6.12: Analysis of activity durations with respect to the image sizes..... 76
Figure 6.13: Time delays for the GPS provisioning services 78
Figure 6.14: Time delays for the GPS provisioning services, shown for the fastest of 10 parallel threads..... 80
Figure 6.15: Time delays for the GPS provisioning services, 10 parallel threads 81
Figure 6.16: Comparison of total request processing time, for GPRS and HSCSD connections 82
..... 82

Figure 7.1: The operational setup of Mobile Host with proxy..... 86
Figure 7.2: SOAP over different protocols..... 87

LIST OF TABLES













Table 2.1: Default data type mapping of kSOAP.....25

Table 4.1: Summary of different possible use cases of Mobile Host.....44

Table 6.1: WS processing % of the total processing time of the request77

List of tables

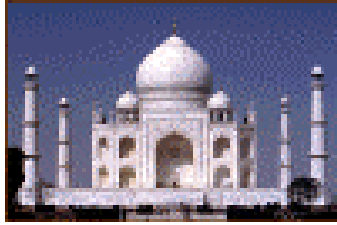
Appendix A - Test case images

 <p>1555.jpg (92.81Kb)</p>	 <p>BabaJaiGuruDeo TempleAgra.jpg (20.72 Kb)</p>	 <p>Kathakali.jpg (3.26 Kb)</p>
 <p>manade1.jpg (7.95 Kb)</p>	 <p>MangrowForest.jpg (46.42 Kb)</p>	 <p>NalandaRuins.jpg (51.29 Kb)</p>
 <p>peacock.jpg (9.74 Kb)</p>	 <p>peee.jpg (23.2 Kb)</p>	 <p>RiceFarming Terraced.jpg (28.36 Kb)</p>
 <p>SanchiGates.jpg (33.35 Kb)</p>	 <p>stigerpond.jpg (15.81 Kb)</p>	 <p>sunderbans8sb.jpg (49.84 Kb)</p>

Appendix A



sunset.jpg (12.91 Kb)



tajmahal.gif (5.45 Kb)



**tajmahal1.gif
(73.77 Kb)**

Appendix B – Some facts and findings

The thesis realized the Mobile Host, where different services can be deployed. For the demonstration of the thesis a standard mobile picture service was developed, that also included the location-based data (GPS data), which provided the exact location (as a map), where the picture was taken. The following subsection gives a brief description of the effort taken.

- The Mobile Host was implemented in PersonalJava, and took approximately 3500 lines of coding (Lok).
- The demo application was developed in swing and took approximately 1000 Lok.
- The utility classes used in the performance analysis were all Java components, requiring approximately 7000 Lok.
- The Mobile Host was tested in real time for more than 100 hrs successfully.
- Approximately 200 data traces were taken under different conditions and were analyzed to reach the experimental results (including all the experiments).
- The Mobile Host was added as an additional component in different projects at Ericsson, where the thesis was performed.
- Further research is being continued at Ericsson research in the domains like Over The Air (OTA) deployment, different possible applications with Mobile Host like Location Based Services (LBS) etc.

Appendix B

BIBLIOGRAPHY

- [1] Steve Graham, Simon Simeonov, Toufic Boubez, Doug Davis, Glen Daniels, Yuichi Nakamura, Ryo Neyama:
Building Web Services with Java
SAMS 2002.
- [2] James Snell, Doug Tidwell, Powell kulchenko:
Programming Web Services with SOAP
O'Reilly, 2002
- [3] James F.Kurose, Keith W. Ross:
Computer Networking, A top-down approach featuring the Internet
Addison Wesley, 2001
- [4] SOAP, Simple Object Access Protocol, version 1.1
<http://www.w3.org/TR/SOAP>
- [5] WSDL, Web Services Description Language, version 1.1
<http://www.w3.org/TR/wsdl>
- [6] HTTP, Hypertext Transfer Protocol version 1.1, IETF RFC 2616
<http://www.ietf.org/rfc/rfc2616.txt>
- [7] “An introduction to J2ME™ development” from Sun™ Tech Days a Developer Conference
- [8] “J2ME building blocks for Mobile Devices”, a white paper on KVM and CLDC by SUN Microsystems
- [9] Micheal Juntao Yuan:
“Access Web services from wireless devices”
Java World online article, August 2002
<http://www.javaworld.com/javaworld/jw-08-2002/jw-0823-wireless.html>
- [10] kXML online documentation at
<http://kxml.enhydra.org/project/aboutProject/index.html>
- [11] A quick introduction to XML at
http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/overview/1_xml.html
- [12] “Java support in Sony Ericsson mobile phones P800 and P802”, January 2003
Developer guidelines from Sony Ericsson Mobile Communications AB,
www.SonyEricssonMobile.com
- [13] Using SOAP in BEEP, IETF RFC 3288
<http://www.ietf.org/rfc/rfc3288.txt>
- [14] kSOAP, a SOAP implementation for Java 2 Microedition
<http://ksoap.enhydra.org/>

Bibliography

- [15] Gzip file format specification, IETF RFC 1952
<http://www.ietf.org/rfc/rfc1952.txt>
- [16] Deflate compressed data format specification, IETF RFC 1951
<http://www.ietf.org/rfc/rfc1951.txt>
- [17] W3Cschool Tutorials
SOAP tutorial
<http://www.w3schools.com/soap/>
- [18] “Quick study on Mobile Web Services Performance” at
<http://research.eed.ericsson.se/projects/>
- [19] Chen & Meixell :
Web Services Enabled Procurement in the Extended Enterprise,
<http://www.csulb.edu/web/journals/jecr/issues/20034/Paper2.pdf>
- [20] Sotamaa, Olli 2002. “All The World's A Botfighters Stage: Notes on Location-Based Multi-User Gaming”. In Frans Mäyrä (ed) Computer Games and Digital Cultures: Conference Proceedings. Tampere: Tampere University Press.